Communication Efficient MPC using Packed Secret Sharing

Vipul Goyal

CMU and NTT Research

Slide credits: Yifan Song

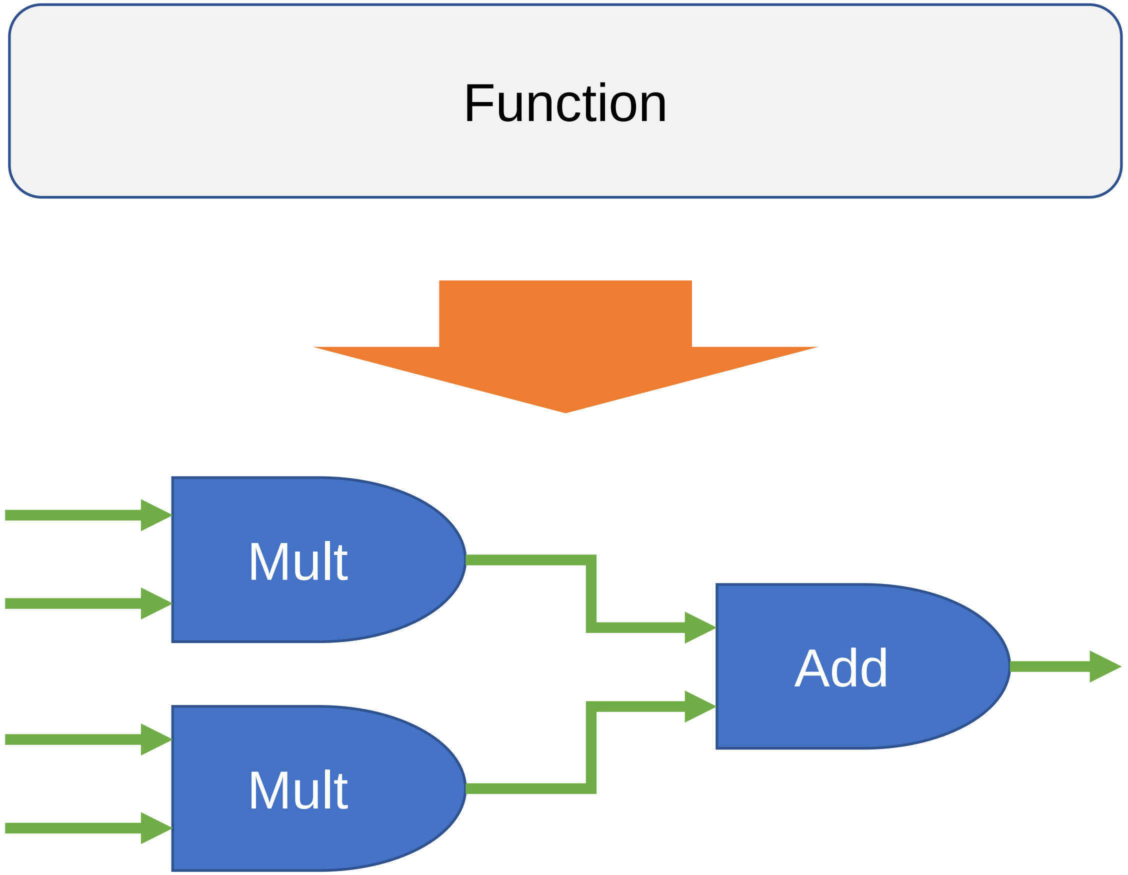
Focus of this Work

- Unconditional Security
  - Honest majority, or dishonest majority in the preprocessing model

- Communication Complexity
  - Key efficiency parameter in the unconditional setting

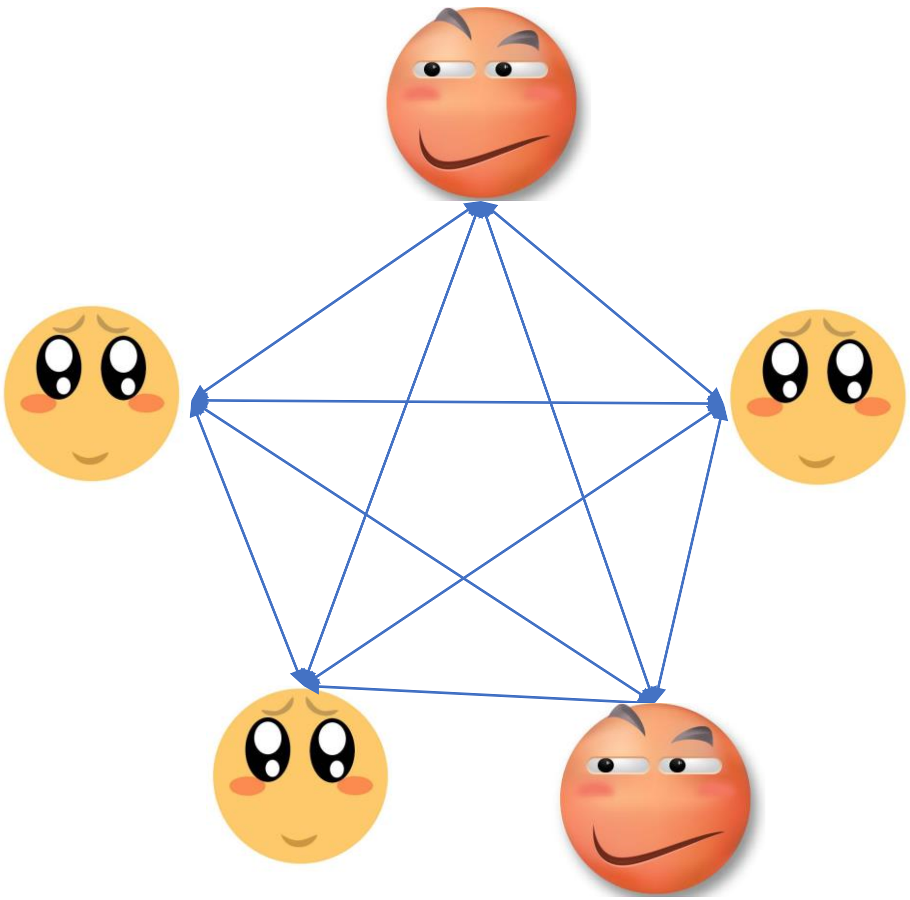
We are interested in constructing communication-efficient information-theoretic MPC

Our Setting



- Function → Arithmetic Circuit (over a finite field)
- Support Additions & Multiplications

Our Setting



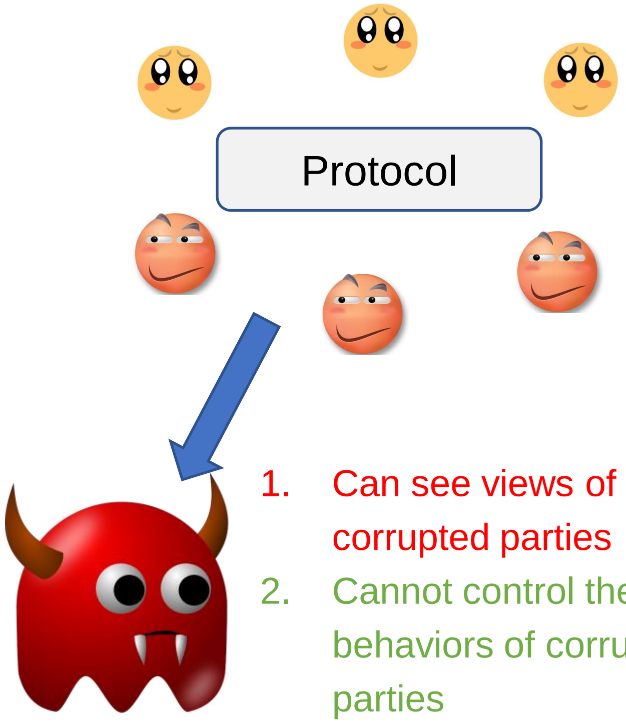
$n = 5$ and $t = 2$

- P2P Channel between every pair of parties.
  - Authenticated, Secure, and Synchronized

- n – number of parties
- t – number of corrupted parties that can be tolerated

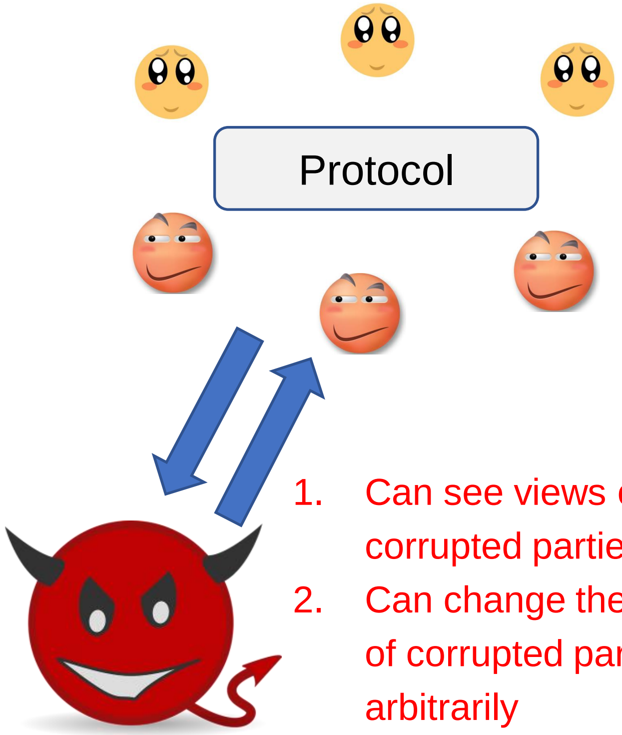
Our Setting – Adversaries

Semi-honest Security



1. Can see views of corrupted parties
2. Cannot control the behaviors of corrupted parties

Malicious Security (with abort)



1. Can see views of corrupted parties
2. Can change the behaviors of corrupted parties arbitrarily

Our Setting – Corruption Threshold

- Common threshold
  - Honest Majority
  - Dishonest Majority (all-but-one corruption)

- Sub-optimal honest majority

$t = (1 - \epsilon) \cdot n/2$ for a constant ϵ

- Sub-optimal dishonest majority

$t = (1 - \epsilon) \cdot n$ for a constant ϵ

Our Works: Sub-optimal Honest Majority

- Goal: use packed secret sharing to obtain asymptotic improvements to CC of MPC

If $t = (1-\epsilon) \cdot n/2$ for a constant ϵ

Overall: O(1/n) field elements per gate per party

(O(C) total)

- Previous works: quite a few, but O(C) was open

Our Works: Sub-optimal Dishonest Majority

- Goal: use packed secret sharing to obtain asymptotic improvements to CC of MPC

If $t = (1 - \epsilon) \cdot n$ for a constant ϵ

Online stage: O(1/n) field elements per gate per party

Preprocessing stage: O(1) field elements per gate per party

Also: Implications to strict honest majority (without preprocessing)

Previous: None in (sub-optimal) dishonest or strict honest majority settings

Why Sub-Optimal Corruption Thresholds?

- Example: MPC on Blockchain
  - 49% is pretty arbitrary. Use 45%?

- Voting, or other settings where we have a large number of parties

Talk Outline

1. **Our Problem and Prior Works**

2. Secret Sharing and the Problem of Sharing Transformation

3. Our Sharing Transformation Construction

4. Using Sharing Transformation to build CE MPC

Background: Honest Majority MPC

- Optimal Threshold Setting ($t = \frac{n-1}{2}$):
  - Best known results achieve $O(n)$ communication per multiplication gate. [DN07, GIP+14, CGH+18, NV18, BBCG+19, GSZ20, BGIN20]
  - The overall communication is $O(|C| \cdot n)$

- Moving to Sub-optimal Threshold Setting ($t = (1/2 - \epsilon) \cdot n$):

Ref.	Circuit Type	Communication
[FY92]	$O(n)$ copies of the same circuit	$O(\|C\|)$
[DIK10, GIP15]	A single circuit	$O(\log\|C\| \cdot \|C\|)$
[GIOZ17]	A single circuit	$O(\log^{1+\epsilon} n \cdot \|C\|)$
[BGJK21]	Restricted Class of Circuits: ***Highly Repetitive Structures***	$O(\|C\|)$

Honest Majority with Sub-Optimal Threshold

Main Theorem [GPS21].

For an arithmetic circuit C, and for all constant $\epsilon > 0$ and $t = (1/2 - \epsilon) \cdot n$, there is an information-theoretic MPC which computes C with $\mathrm{O}(|\mathrm{C}|)$ communication.

Example Corollary: For $t = 0.49n$, the communication complexity is $\mathrm{O}(|\mathrm{C}|)$.

Hence: as number of parties go up, communication per party goes down

Honest Majority with Sub-Optimal Threshold

Main Theorem [GPS21].

For an arithmetic circuit C, and for all constant $\epsilon > 0$ and $t = (1/2 - \epsilon) \cdot n$, there is an information-theoretic MPC which computes C with $O(|C|)$ communication.

A factor of $O(n)$ improvement

compared with protocols in the standard honest majority setting.

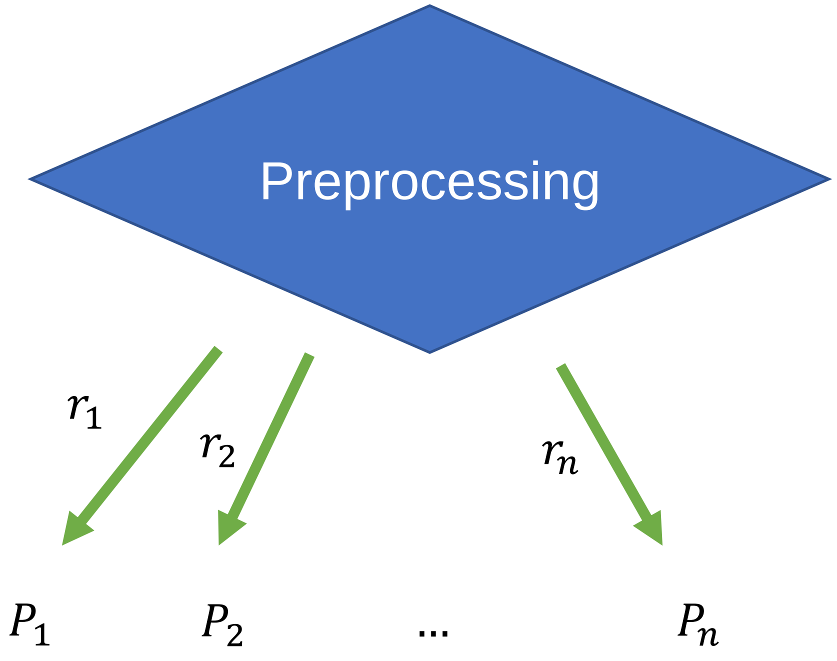
Moving to Dishonest Majority

- Negative result [BGW88]:

“Information-theoretic MPC cannot exist without honest majority”

- To overcome:

Circuit-Independent Preprocessing Phase

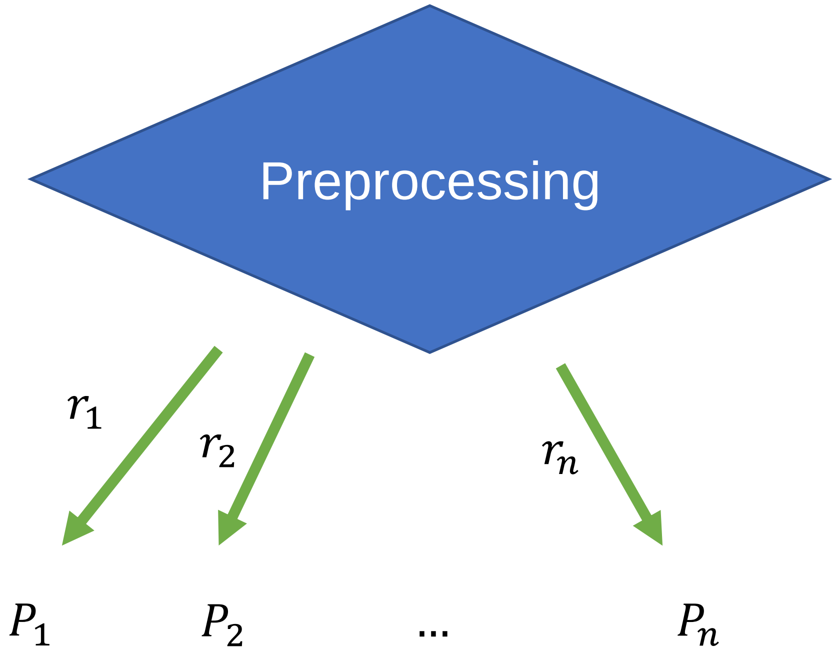


Moving to Dishonest Majority

- Feasible Results
  - [Kil88, IPS08] ⇒ IT MPC is possible
- Well-Known Result [DPSZ12]
  - All-but-one corruption setting
  - The cost is

Preprocessing Data	Online Communication
$O(\lvert C\rvert \cdot n)$	$O(\lvert C\rvert \cdot n)$

Circuit Independent Preprocessing Phase



Our Results [GPS22]

Main Theorem --- Semi-Honest (*Informal*).

For an arithmetic circuit C over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq |\mathrm{C}| + n$, and for all constant $\epsilon > 0$ and $t = (1-\epsilon) \cdot n$, there is a semi-honest IT MPC which computes C with $O(|\mathrm{C}|)$ elements of both preprocessing data and communication complexity.

Techniques

- Efficient Sharing Transformation Protocols.
- Sparsely Packed Shamir Secret Sharing Scheme, packed Beaver Triples.

Also: extension to malicious security, smaller fields (later)

Implication to (Strict) Honest Majority

Corollary --- IT MPC with Honest Majority.

- $O(|C|)$ Communication Complexity for the Online Phase
- $O(|C| \cdot n)$ Communication Complexity for the Offline Phase

Techniques

- Set $\epsilon = 1/2$ in our main theorem.
- Use the state-of-the-art MPC protocol for honest majority to instantiate the preprocessing phase.

Implication to (Strict) Honest Majority

Corollary --- IT MPC with Honest Majority.

- $O(|C|)$ Communication Complexity for the Online Phase
- $O(|C| \cdot n)$ Communication Complexity for the Offline Phase

The First Result in the Honest Majority Setting that

- Achieves sub-linear communication complexity in the # parties in the online phase,
- Maintains $O(|C| \cdot n)$ overall asymptotic communication complexity.

Other Related Works

- Trading Offline Preprocessing with Online Communication
  (All-but-one Corruption Setting)
  - [IKM+13] --- Can Achieve Linear C.C. Only in the Input Size.
    Require Exponential Pre Data in the Circuit Size.
  - [Cou19] --- Exploring a Balance between Preprocessing and Communication

Online Communication	Offline Preprocessing
$\|C\| \cdot n/k$	$\|C\| \cdot n \cdot 2^{k+2^{2^k}}/k$

Talk Outline

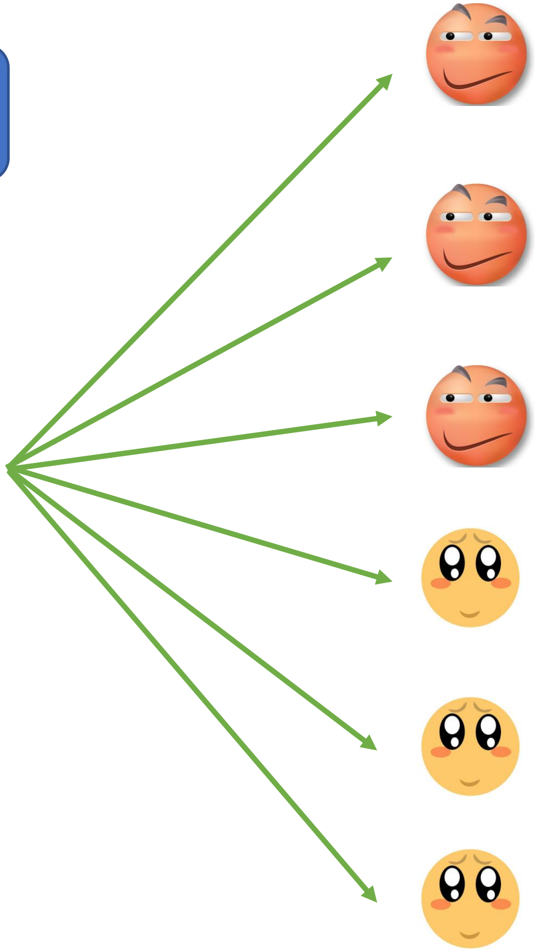
1. Our Problem and Prior Works
2. **<u>Secret Sharing and the Problem of Sharing Transformation</u>**
3. Our Sharing Transformation Construction
4. Using Sharing Transformation to build CE MPC

Secret Sharing

Can be a single value or a vector

Secret: s

Threshold: t, t'



share_1

share_2

share_3

share_4

share_5

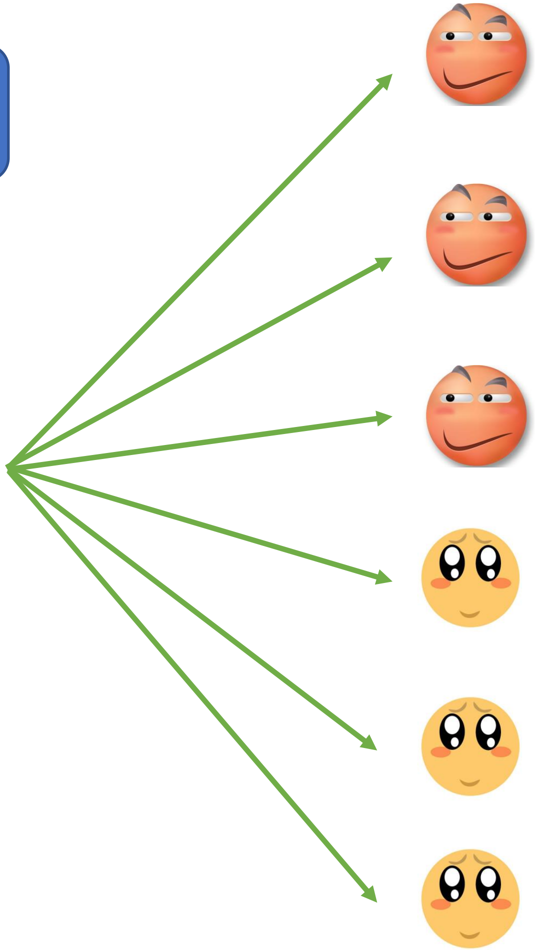
share_6

Secret Sharing

Can be a single value or a vector

Secret: s

Threshold: t, t'



share_1

share_2

share_3

share_4

share_5

share_6

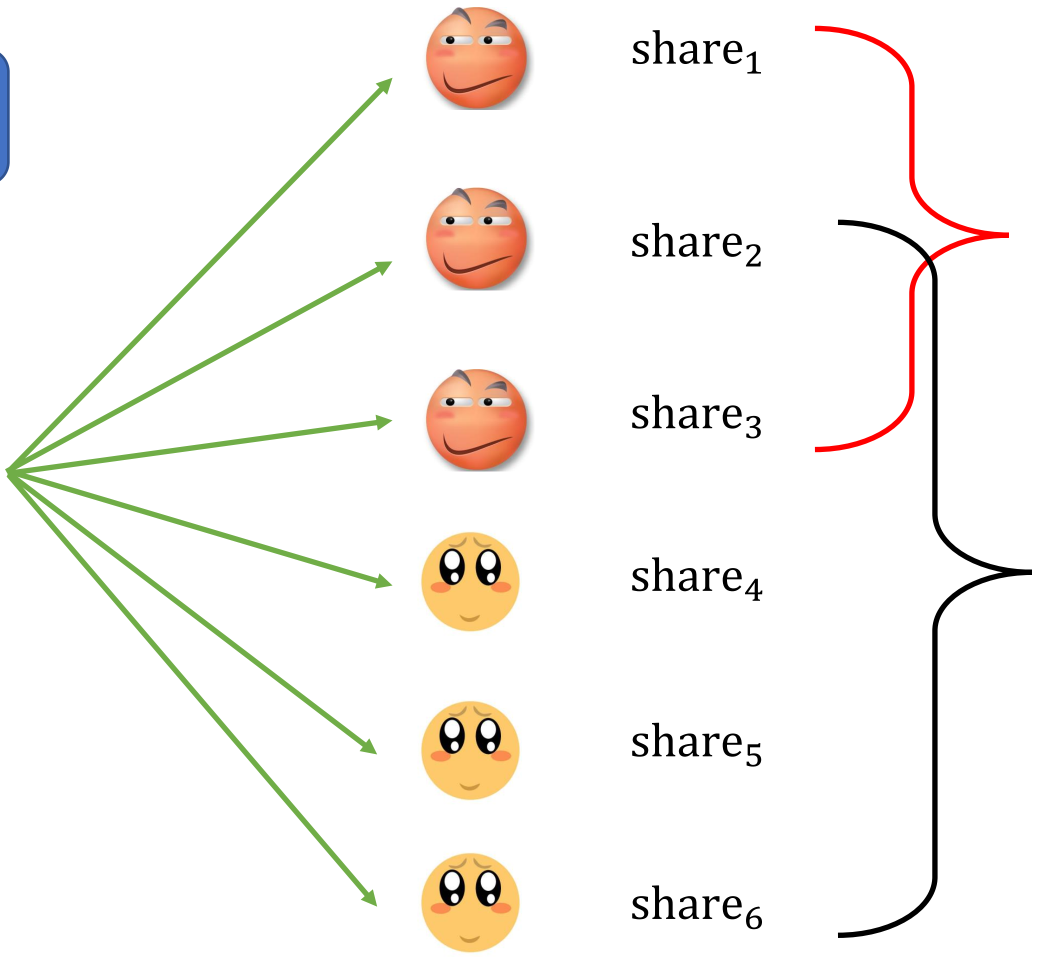
Any t shares together reveal no information about the secret s.

Secret Sharing

Can be a single value or a vector

Secret: s

Threshold: t, t'



Any t shares together reveal no information about the secret s.

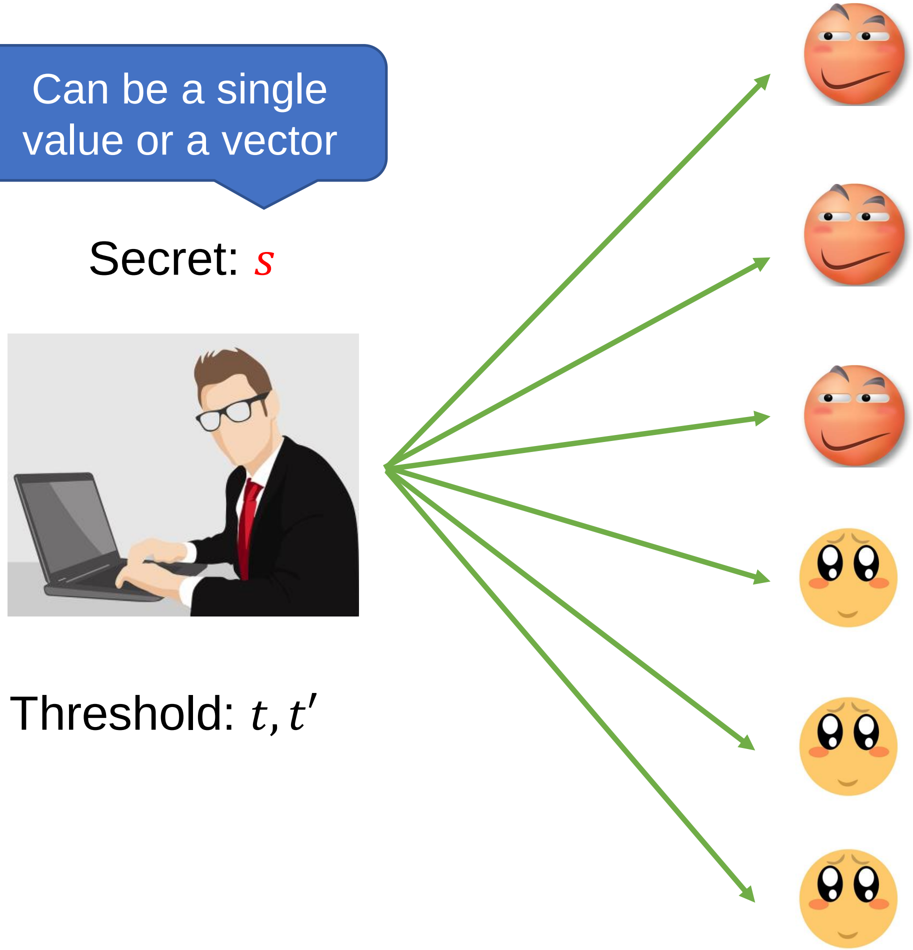
Any t' shares can reconstruct the secret s.

Secret Sharing

Convention: Bold font denotes vector of secrets $\boldsymbol{X}$

Can be a single value or a vector

Secret: s



Threshold: t, t'

share_1

share_2

share_3

share_4

share_5

share_6

Denoted by X
(or $[s]$)

Linear Secret Sharing

- A Secret Sharing Scheme is Linear if

$$\begin{array}{ccccc} \boldsymbol{X} & + & \boldsymbol{Y} & \Rightarrow & \boldsymbol{Z} \\ \big\downarrow & & \big\downarrow & & \big\downarrow \\ s_1 & & s_2 & & s_1 + s_2 \end{array}$$

Sharing Transformation

- Given two linear secret sharing schemes and a linear function *in the same finite field*:

$$(\Sigma, \Sigma', f(\cdot))$$

- All parties hold a Σ-sharing $\boldsymbol{X}$. The goal is to obtain a Σ'-sharing $\boldsymbol{X}'$

$$\begin{array}{ccc} \boldsymbol{X} & \Rightarrow & \boldsymbol{X}' \\ \downarrow & & \downarrow \\ x & \Rightarrow & f(x) \end{array}$$

Example 1: Degree Reduction

- MPC Over Large Fields: [BGW88, DN07]
  - For Multiplication Gates --- Can locally compute result *but in a different secret sharing scheme*
  - **Need to Transform the Result to the Original Secret Sharing Scheme**

$$[x], [y] \xrightarrow[\text{Computation}]{\text{Local}} \langle x \cdot y \rangle \xrightarrow[\text{Transformation}]{\text{Sharing}} [x \cdot y]$$

Example 2: RMFE

- MPC Over Small Fields: [CCXY18, PS21, CRX21]
  - Use Reverse Multiplication Friendly Embeddings (RMFE) to transform to computation over large fields but *resulting in the secrets encoded in a different form*
  - **Need to Transform the Result by Using the Original Encoding Scheme**

Outputs are encoded by ψ

$$[\phi(\boldsymbol{x})],[\phi(\boldsymbol{y})] \xrightarrow[\text{Protocol}]{\text{Multiplication}} [\psi(\boldsymbol{x}*\boldsymbol{y})]=[\phi(\boldsymbol{x})\cdot\phi(\boldsymbol{y})]$$

Inputs are encoded by ϕ

$$[\psi(\boldsymbol{x}*\boldsymbol{y})] \xrightarrow[\text{Transformation}]{\text{Sharing}} [\phi(\boldsymbol{x}*\boldsymbol{y})]$$

Example 3: Network Routing (Our Focus)

- MPC via Packed Secret Sharings: [DIK10,GIP15,GSY21,BGJK21,GPS21]
  - Use packed Shamir sharings to evaluate a single circuit.
  - Main difficulty --- *Network Routing*
  - **Need to Perform Linear Map on the Secrets of a Single Sharing.**

$$[x_1, x_2, x_3] \xrightarrow{\text{Permutation}} [x_2, x_3, x_1]$$

$$[x_1, x_2, x_3] \xrightarrow{\text{Fan-out}} [\underline{\underline{x_1, x_1}}, x_3]$$

Our Question: Sharing Transformation in Batches

- Sharing Transformation Occurs Frequently in MPC

- Previous solutions are efficient (linear in the number of parties) when *the same sharing transformation* is performed many times.
  - This is sufficient for the first two examples
  - But not for the third example: Different permutations (or different pattern of fan-out) corresponds to different sharing transformations

Our Question

- Sharing Transformation Occurs Frequently in MPC

Can we achieve linear communication complexity

in the number of parties for different sharing transformations?

Our Result

Theorem --- Sharing Transformation (*Informal*).

Let $k = n - t$. For all $\{(\Sigma_i, \Sigma_i', f_i)\}_{i=1}^{k}$, there is an efficient protocol against t corrupted parties, which transforms

$$\begin{array}{ccc} \boldsymbol{X}_i \in \Sigma_i & \Rightarrow & \boldsymbol{X'}_i \in \Sigma_i' \\ \big\downarrow & & \big\downarrow \\ x_i & \Rightarrow & x_i' = f_i(x_i) \end{array}$$

The achieved communication complexity per transformation is $O(n^3/k^2)$, which is $O(n)$ when $k = O(n)$.

(Cost grows linearly with the share size

For share size ℓ elements → $O(n \cdot \ell)$ elements)

Our Result

- A Generic Approach for Sharing Transformation
  - Work for all linear sharing transformations
  - Achieve linear communication complexity in the number of parties
  - Linear cost can be achieved for different sharing transformations

- Enable A **New Approach** for MPC with Sub-optimal Corruption Threshold

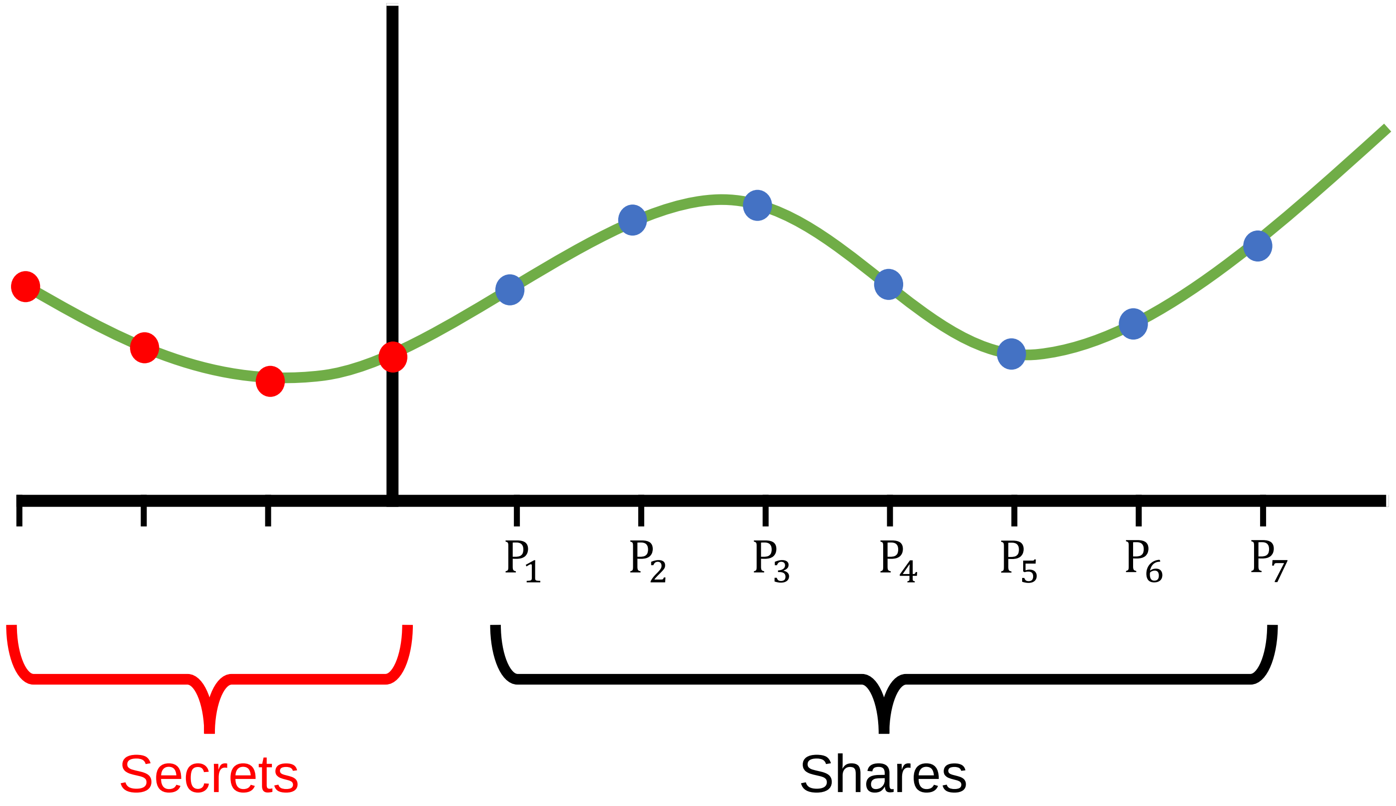
Talk Outline

1. Our Problem and Prior Works
2. Secret Sharing and the Problem of Sharing Transformation
3. **<u>Our Sharing Transformation Construction</u>**
4. Using Sharing Transformation to build CE MPC

Packed Shamir Secret Sharing

Secrets:

$s_1, s_2, \dots, s_k$



Packed Shamir Secret Sharing

- Properties:
  1. Linear Homomorphism:

$$[\boldsymbol{x} + \boldsymbol{y}] = [\boldsymbol{x}] + [\boldsymbol{y}]$$

Follow from the addition of underlying two polynomials.

  2. Multiplication:

$$[\boldsymbol{x} * \boldsymbol{y}] = [\boldsymbol{x}] * [\boldsymbol{y}]$$

Follow from the multiplication of underlying two polynomials.

Degree Increases

Omitted for Simplicity

Overview of Construction

1. Reduce to preparing sharings of random secrets
2. Prepare a single sharing
3. Prepare a batch of sharings

Reduce to Preparing Random Sharings

- Given $(\Sigma, \Sigma', f(\cdot))$

$$\begin{array}{ccc} \boldsymbol{X} \in \Sigma & \Rightarrow & \boldsymbol{X}' \in \Sigma' \\ \big\downarrow & & \big\downarrow \\ x & \Rightarrow & f(x) \end{array}$$

Reduce to Preparing Random Sharings

- Given $(\Sigma, \Sigma', f(\cdot))$

$$\begin{array}{ccc} \boldsymbol{X} \in \Sigma & \Rightarrow & \boldsymbol{X}' \in \Sigma' \\ \downarrow & & \downarrow \\ x & \Rightarrow & f(x) \end{array}$$

- Sufficient to prepare $(\boldsymbol{R}, \boldsymbol{R}')$ [DIK10]

$$\begin{array}{ccc} \boldsymbol{R} \in \Sigma & \Rightarrow & \boldsymbol{R}' \in \Sigma' \\ \downarrow & & \downarrow \\ r & \Rightarrow & f(r) \end{array}$$

Reduce to Preparing Random Sharings

- Given X and R, parties can locally compute shares of x+r and send to P1
- P1 reconstructs x+r, applies the linear transformation f and sends f(x+r) to all parties
- Given R’, parties can get X’: parties locally compute shares of f(x+r) – r’:

$$f(x+r) - r' = f(x) + f(r) - r' = f(x) + r' - r = f(x)$$

From Two Sharings to One Sharing

- Observe that $(\Sigma, \Sigma', f(\cdot))$ itself defines a *single* linear secret sharing scheme Π
- The goal is to prepare a random sharing $(\boldsymbol{R}, \boldsymbol{R}')$ in Π

How to Prepare Random Sharings?

- Given a linear secret sharing scheme Π, the goal is to prepare a random sharing in Π

- Previous solutions based on [DN07] require to prepare $O(n)$ random sharings to be efficient
  - $\Rightarrow O(n^2)$ communication complexity for $O(n)$ random sharings

Our Goal

- Prepare $O(n)$ random sharings, *each for a different secret sharing scheme*

- Linear secret sharing schemes:

Each share is a linear combination of the inputs

$$\mathrm{Share}_{\Pi}(s_1, \dots, s_\ell; r_1, \dots, r_m) = (\mathrm{sh}_1, \mathrm{sh}_2, \dots, \mathrm{sh}_n)$$

Our Solution

Our Idea

- View Share_{Π} as a circuit (only contain addition gates)
- Compute Share_{Π} via an MPC protocol

Our Idea (Continued)

1. Prepare a random Shamir sharing for each s_i and r_i
2. Locally compute a sharing of sh_j and reconstruct it to P_j

- The common point of linear secret sharing schemes:

Each share is a linear combination of the inputs

$$\mathrm{Share}_{\Pi}(s_1, \ldots, s_\ell; r_1, \ldots, r_m) = (\mathrm{sh}_1, \mathrm{sh}_2, \ldots, \mathrm{sh}_n)$$

Our Solution

Π:

s_1	s_2	r_1	r_2	r_3	r_4	$\rightarrow$	sh_1	sh_1	...	sh_n

$[\gamma_1]$	$[\gamma_2]$	$[\gamma_3]$	$[\gamma_4]$	$[\gamma_5]$	$[\gamma_6]$	$\rightarrow$	$[\mathrm{sh}_1]$	$[\mathrm{sh}_2]$	...	$[\mathrm{sh}_n]$

A Random Sharing for Each Column

Reconstruct $[\mathrm{sh}_j]$ to P_j

Our Solution

Our Idea

- View Share_Π as a circuit (only contain addition gates)
- Compute Share_Π via an MPC protocol

Drawback

- Require to prepare $\ell + m$ random Shamir sharings
- Require $O(n^2)$ communication complexity
- Both prep data and C.C. are $O(n^2)$

- The common point of linear secret sharing schemes:

Each share is a linear combination of the inputs

$$\mathrm{Share}_\Pi(s_1, \ldots, s_\ell; r_1, \ldots, r_m) = (\mathrm{sh}_1, \mathrm{sh}_2, \ldots, \mathrm{sh}_n)$$

Our Solution

- Prepare $O(n)$ random sharings, *each for a different secret sharing scheme*

Our Idea

- View Share_{Π} as a circuit (only contain addition gates)
- Compute Share_{Π} via an MPC protocol
- Pack the computation for $k = O(n)$ different sharing schemes $\Pi_1, \Pi_2, \ldots, \Pi_k$

Main Observation

For Π_i and Π_j,

$$\mathrm{Share}_{\Pi_i} \text{ and } \mathrm{Share}_{\Pi_\mathrm{j}}$$

have *the same structure* but *use different coefficients*.

Our Solution

Π_1:	$s_1^{(1)}$	$s_2^{(1)}$	$s_3^{(1)}$	$r_1^{(1)}$	$r_2^{(1)}$	$r_3^{(1)}$	$\rightarrow$	$\mathrm{sh}_1^{(1)}$	$\mathrm{sh}_2^{(1)}$	…	$\mathrm{sh}_n^{(1)}$
Π_2:	$s_1^{(2)}$	$s_2^{(2)}$	$r_1^{(2)}$	$r_2^{(2)}$	$r_3^{(2)}$	$r_4^{(2)}$	$\rightarrow$	$\mathrm{sh}_1^{(2)}$	$\mathrm{sh}_2^{(2)}$	…	$\mathrm{sh}_n^{(2)}$
Π_3:	$s_1^{(3)}$	$s_2^{(3)}$	$s_3^{(3)}$	$s_4^{(3)}$	$r_1^{(3)}$	$r_2^{(3)}$	$\rightarrow$	$\mathrm{sh}_1^{(3)}$	$\mathrm{sh}_2^{(3)}$	…	$\mathrm{sh}_n^{(3)}$
	$[\gamma_1]$	$[\gamma_2]$	$[\gamma_3]$	$[\gamma_4]$	$[\gamma_5]$	$[\gamma_6]$	$\rightarrow$	$[\mathbf{sh}_1]$	$[\mathbf{sh}_2]$	…	$[\mathbf{sh}_n]$

A Random Packed Sharing for Each Column

Reconstruct $[\mathbf{sh}_j]$ to P_j

Our Solution Contd..

Π_1:	$s_1^{(1)}$	$s_2^{(1)}$	$s_3^{(1)}$	$r_1^{(1)}$	$r_2^{(1)}$	$r_3^{(1)}$	$\rightarrow$	$\mathrm{sh}_1^{(1)}$	$\mathrm{sh}_2^{(1)}$	…	$\mathrm{sh}_n^{(1)}$
Π_2:	$s_1^{(2)}$	$s_2^{(2)}$	$r_1^{(2)}$	$r_2^{(2)}$	$r_3^{(2)}$	$r_4^{(2)}$	$\rightarrow$	$\mathrm{sh}_1^{(2)}$	$\mathrm{sh}_2^{(2)}$	…	$\mathrm{sh}_n^{(2)}$
Π_3:	$s_1^{(3)}$	$s_2^{(3)}$	$s_3^{(3)}$	$s_4^{(3)}$	$r_1^{(3)}$	$r_2^{(3)}$	$\rightarrow$	$\mathrm{sh}_1^{(3)}$	$\mathrm{sh}_2^{(3)}$	…	$\mathrm{sh}_n^{(3)}$
	$[\gamma_1]$	$[\gamma_2]$	$[\gamma_3]$	$[\gamma_4]$	$[\gamma_5]$	$[\gamma_6]$	$\rightarrow$	$[\mathbf{sh}_1]$	$[\mathbf{sh}_2]$	…	$[\mathbf{sh}_n]$

A Random Packed Sharing for Each Column

Reconstruct $[\mathbf{sh}_j]$ to P_j

Our Solution

- Prepare $O(n)$ random sharings, *each for a different secret sharing scheme*

Our Idea

- View Share_{Π} as a circuit (only contain addition gates)
- Compute Share_{Π} via an MPC protocol
- Pack the computation for $k = O(n)$ different sharing schemes $\Pi_1, \Pi_2, \ldots, \Pi_k$

Our Result

- Prepare $k = O(n)$ sharings each time
- Prep Data: $O(n^2)$ elements
- Communication: $O(n^2)$ elements
- Amortized cost: $O(n)$ elements

Talk Outline

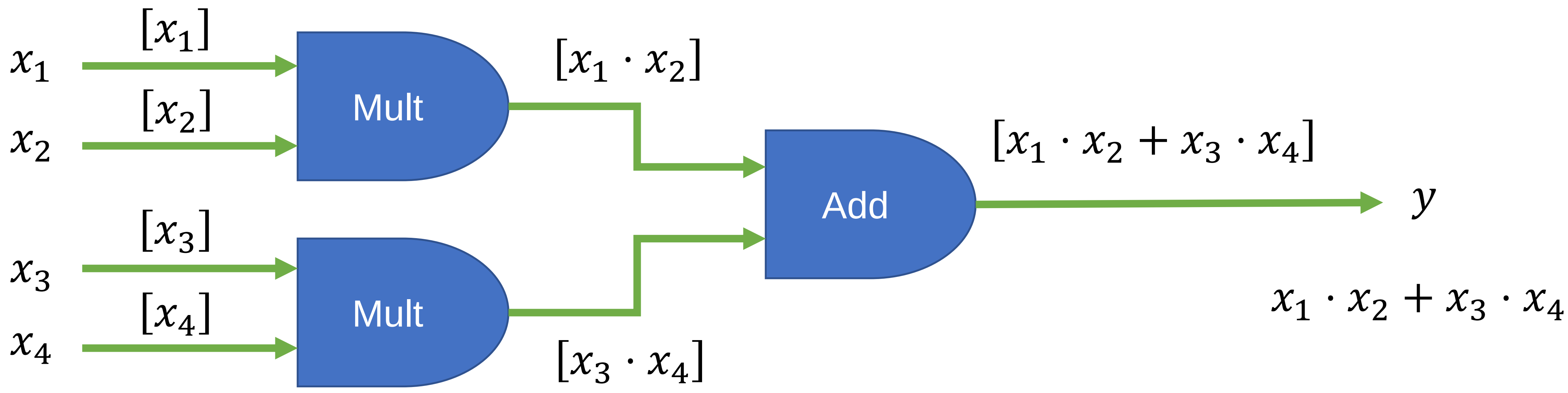
1. Our Problem and Prior Works

2. Secret Sharing and the Problem of Sharing Transformation

3. Our Sharing Transformation Construction

4. **Using Sharing Transformation to build CE MPC**

Designing MPC



High-Level Idea

For each wire, compute a secret sharing of the value carried by this wire.

Problem is reduced to *evaluate addition and multiplication gates*.

Sub-optimal Corruption Threshold

- Use the Packed Secret Sharing Technique [FY92]
  - Replace $[x_1], [x_2], \ldots, [x_k]$ by $[x_1, x_2, \ldots, x_k]$. ($k \Leftarrow$ packing parameter)

k individual sharings

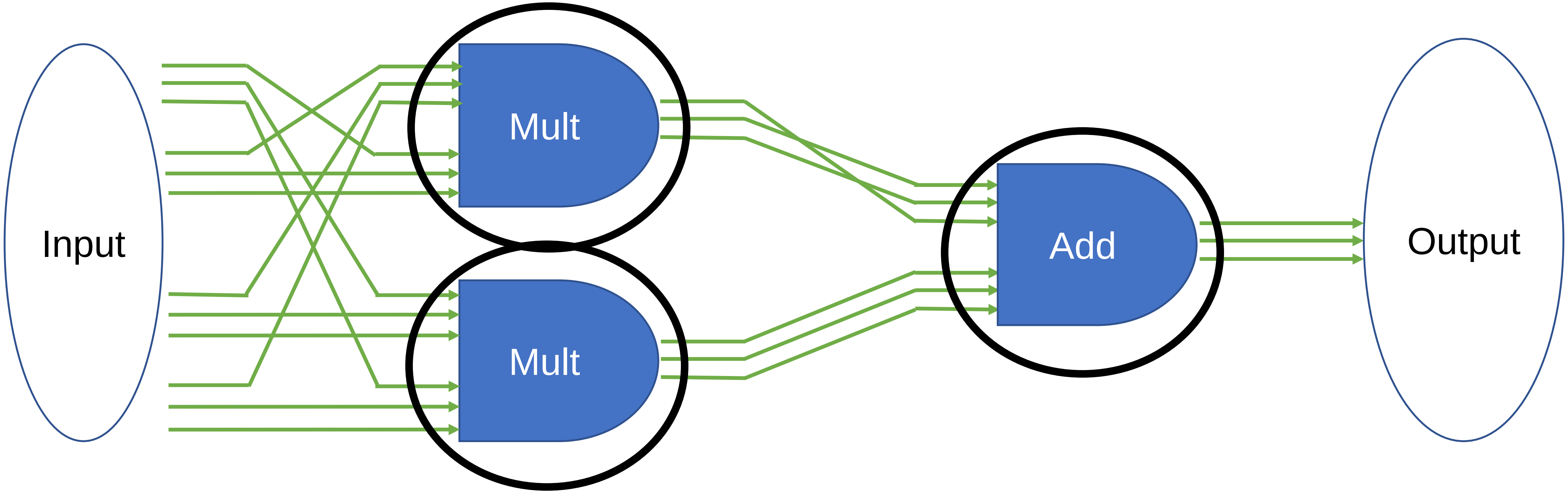
1 packed sharing

  - Compute $\mathrm{OP} \in \{\mathrm{Mult}, \mathrm{Add}\}$ of two packed sharings at cost 1

$$\mathrm{OP}([x_1, x_2, \ldots, x_k], [y_1, y_2, \ldots, y_k]) = [z_1, z_2, \ldots, z_k]$$

  - Ideally, the *cost per gate* is reduced by a factor of k

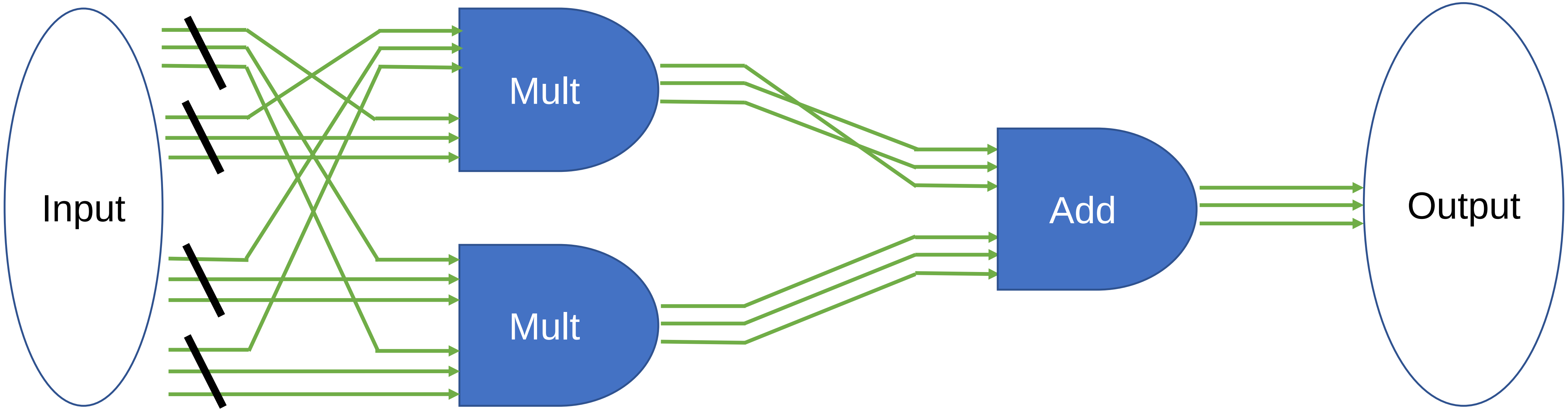
General Circuit via Packed Sharing



High-Level Idea 1

Group Gates of the Same Type in Each Layer

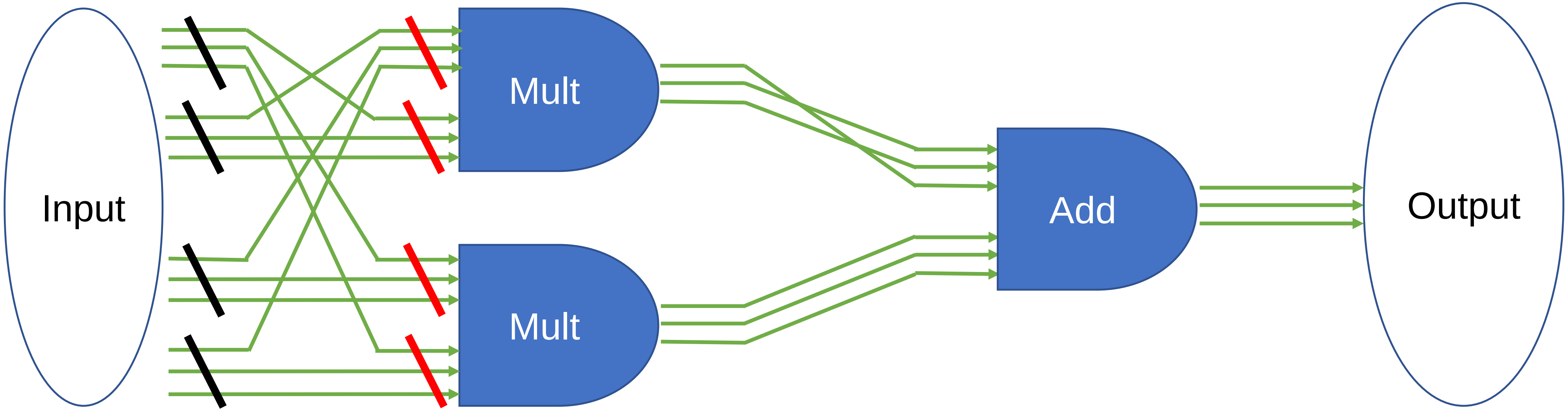
General Circuit via Packed Sharing



High-Level Idea 2

Each Party Shares its Inputs via Packed Sharings

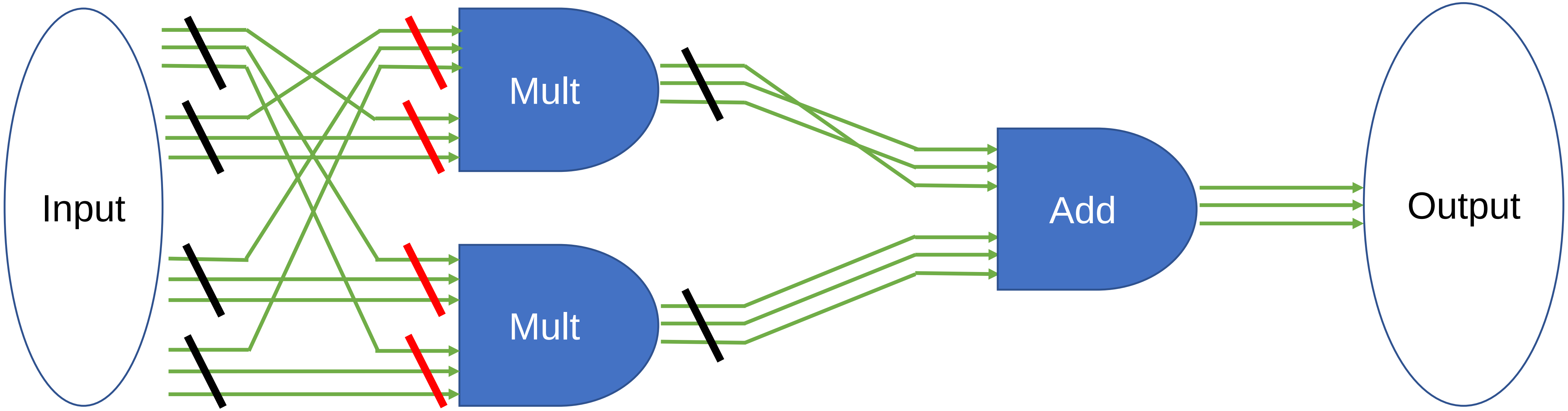
General Circuit via Packed Sharing



High-Level Idea 3

Compute Input Packed Sharings for Each Group of Gates

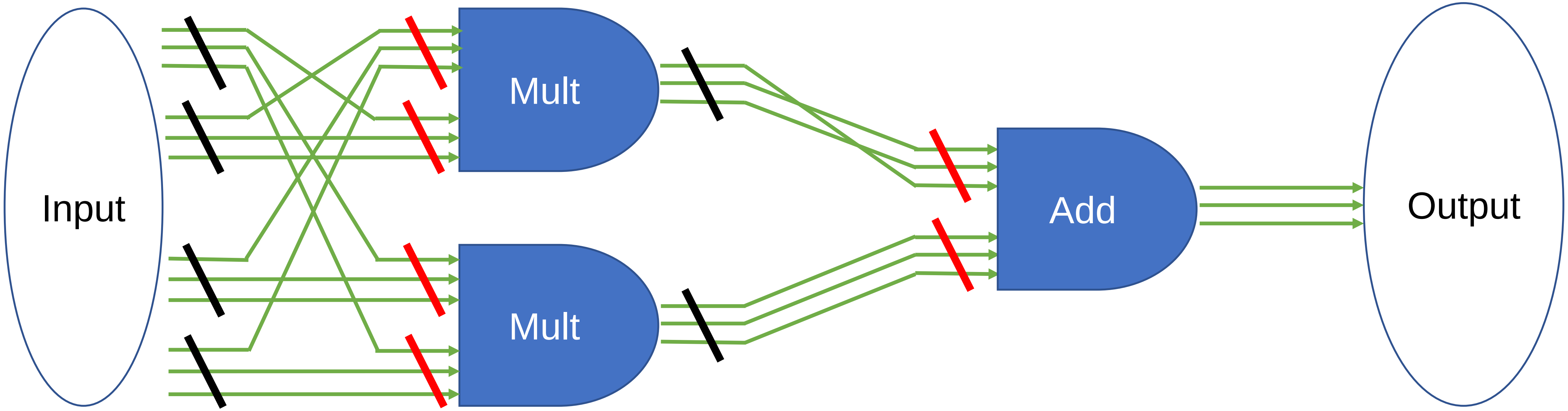
General Circuit via Packed Sharing



High-Level Idea 4

Evaluate Each Group of Gates at Cost 1

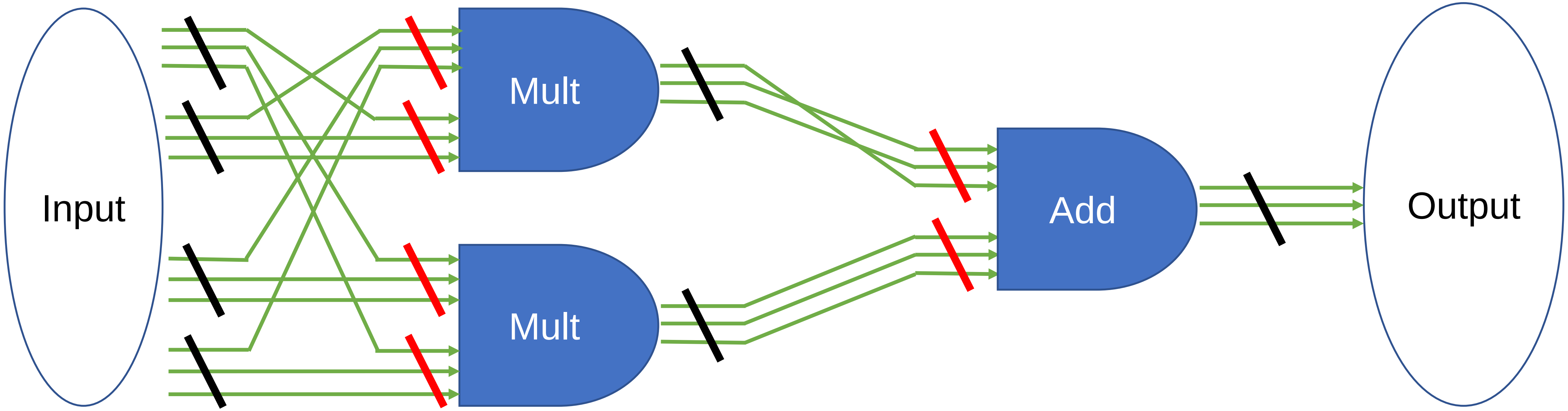
General Circuit via Packed Sharing



High-Level Idea 4

Evaluate Each Group of Gates at Cost 1

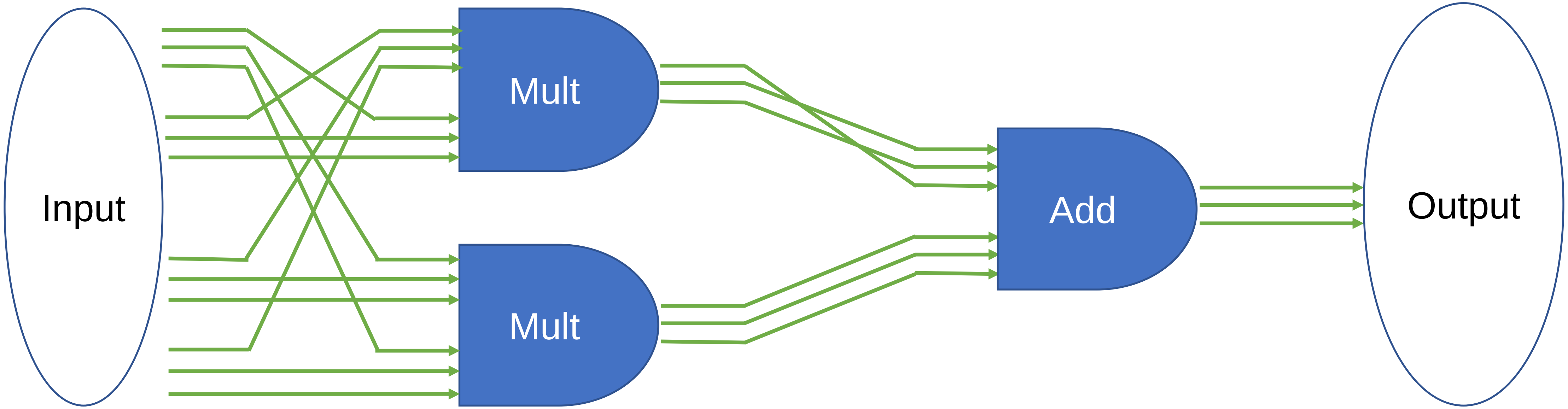
General Circuit via Packed Sharing



High-Level Idea 5

Reconstruct Outputs

Main Difficulty: Network Routing

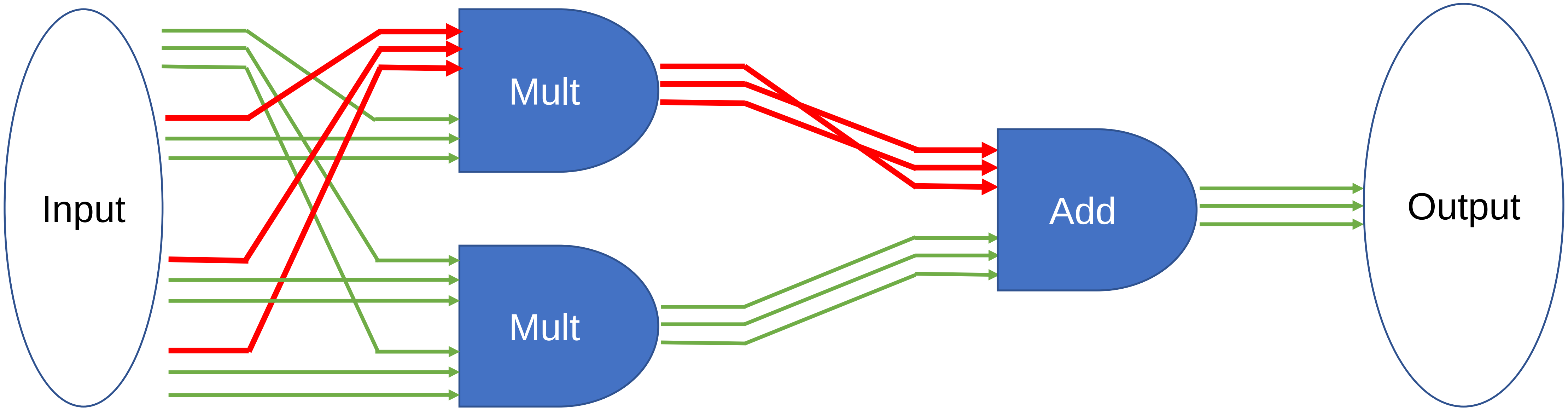


How should we

prepare input packed sharings of the current layer

from output packed sharings in previous layers?

Main Difficulty: Network Routing



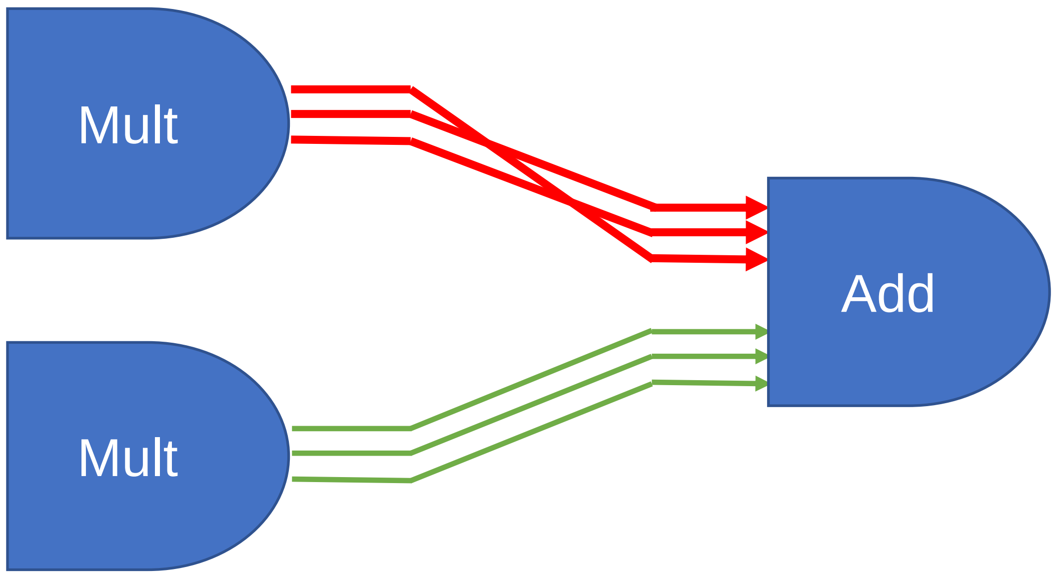
How should we

prepare input packed sharings of the current layer

from output packed sharings in previous layers?

Main Difficulty: Network Routing

- Difficulty 1: Secret Reordering



Solved Directly by

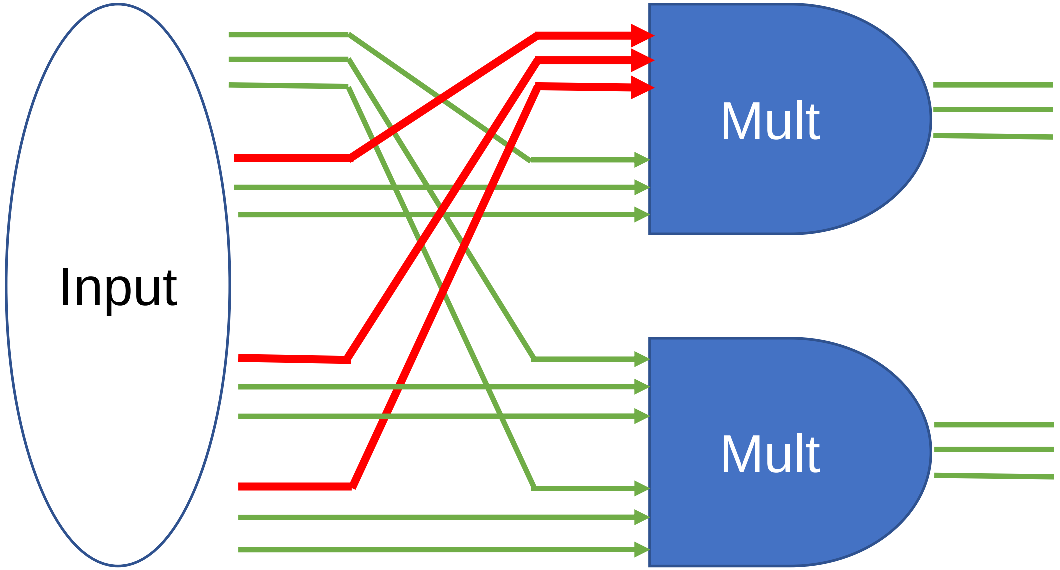
Sharing Transformation

$[x_1, x_2, x_3] \longrightarrow [x_2, x_3, x_1]$

Main Difficulty: Network Routing

- Difficulty 2: Secret Collection

Unclear how to directly use sharing transformation. Results in $O(k \cdot n)$ communication



$$[y_1, y_2, y_3] \quad [z_1, z_2, z_3] \quad [w_1, w_2, w_3] \longrightarrow [y_1, z_1, w_1]$$

Network Routing: Solution in [GPS21]

- Efficient Secret Collection for Restricted Cases

$[x_1, x_2, x_3]$ $\quad$ $[y_1, y_2, y_3]$ $\quad$ $[z_1, z_2, z_3]$

$$\Downarrow$$

$$[x_1, y_2, z_3]$$

Network Routing: Solution in [GPS21]

- Efficient Secret Collection for Restricted Cases

$$[x_1, x_2, x_3] * [1,0,0] = [x_1, 0,0]$$

$$[y_1, y_2, y_3] * [0,1,0] = [0, y_2, 0]$$

$$[z_1, z_2, z_3] * [0,0,1] = [0,0, z_3]$$

$$[x_1, y_2, z_3]$$

Network Routing: Solution in [GPS21]

- x_1, x_2, x_3 are points on a polynomial

- Choose another polynomial where the corresponding points are 1, 0, 0.

- Multiply the two polynomials: corresponding points on the resulting polynomial are $x_1, 0{,}0$

- Degree goes up but can be reduced

$$\boxed{\begin{array}{c}[x_1, x_2, x_3] \\ * \\ [1,0,0] \\ = \\ [x_1, 0{,}0]\end{array}}$$

Are We Done?

- If secrets are stored at the same positions…

$$[x_1, x_2, x_3] * [1,0,0] = [x_1, 0,0]$$

$$[y_1, y_2, y_3] * [1,0,0] = [y_1, 0,0]$$

$$[z_1, z_2, z_3] * [1,0,0] = [z_1, 0,0]$$

$$[x_1, y_1, z_1]$$

Non-Collision Property [GPS21]

- Non-collision Property

  For all sharing, secrets we need to collect all come from different positions

- To Achieve Non-collision Property

  1. Need to compile the circuit, perform proper Fan-out operations and Permutations on each packed sharing
  2. Existence of Permutations relies on Hall's Marriage Theorem

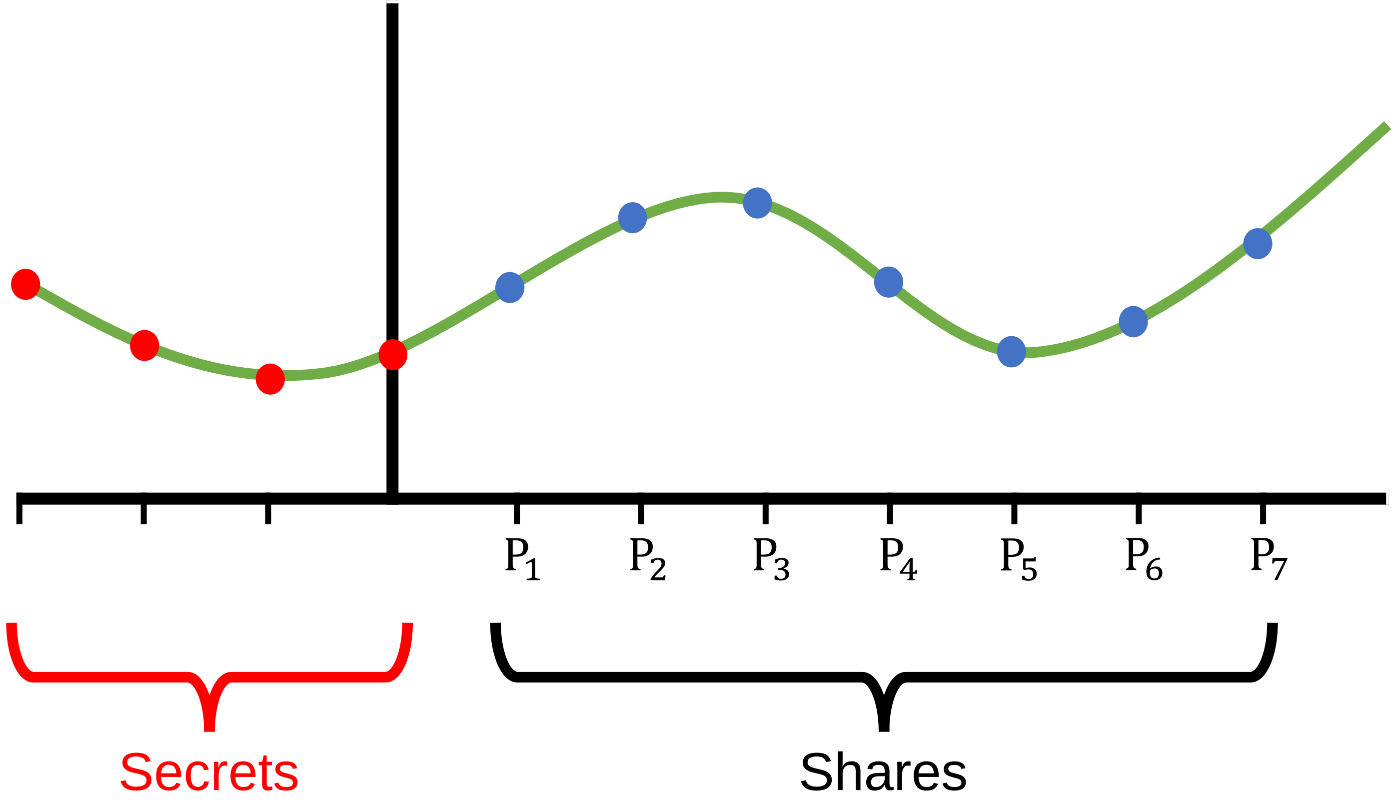
Goal: Ensuring No-Collision Property

Goal: For each sharing, secrets we need to collect all come from different positions in previous sharings

Tool: Sparsely Packed Shamir Sharings

Packed Shamir Secret Sharing

Secrets:
$s_1, s_2, \ldots, s_k$



Packed Shamir Sharings

A polynomial $f(\cdot)$ --- Packing Parameter $k = 3$

α_1	α_2	…	α_n		1	2	3	4	5	…	m

sh_1	sh_2	…	sh_n		s_1	s_2	s_3	$\cdot$	$\cdot$	…	$\cdot$

$$[\ s_1, s_2, s_3 \parallel 1, 2, 3\ ]$$

Packed Shamir Sharings

- We could store secrets differently. Why?

A polynomial $f(\cdot)$ --- Packing Parameter $k = 3$

α_1	α_2	…	α_n		1	2	3	4	5	…	m
sh_1	sh_2	…	sh_n		s_1	$\cdot$	s_2	$\cdot$	s_3	…	$\cdot$

$$[\ s_1, s_2, s_3 \parallel 1, 3, 5\ ]$$

Sparsely Packed Shamir Sharings

- Sparsely Packed Shamir Sharings

Different Sharings store secrets in different locations (big Field)

- Suppose the field size $|\mathrm{F}| \geq |C| + n$

Use a different set of positions for each group of gates

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

Back to Network Routing

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

Observation

- Secrets are stored at different positions.
- Non-collision property is achieved for free.

[GPS21] Achieving non-collision property was expensive

Example: Network Routing

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

- Suppose we want to collect secrets x_1, y_1, z_1, and store them at $(10,11,12)$

Example: Network Routing

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

- Suppose we want to collect secrets x_1, y_1, z_1, and store them at $(10,11,12)$

- **Step 1: Locally compute** $[\ x_1, y_1, z_1 \parallel 1, 4, 7\ ]$

How?

Example: Network Routing

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$
$\downarrow$	$\downarrow$	$\downarrow$
$[\ x_1, \cdot, \cdot \parallel 1, 4, 7\ ]$	$[\ \cdot, y_1, \cdot \parallel 1, 4, 7\ ]$	$[\ \cdot, \cdot, z_1 \parallel 1, 4, 7\ ]$

All three sharings can be viewed as sharings that uses positions 1, 4, and 7 (only care about x_1, y_1, z_1)

Example: Network Routing

Group 1

$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$

Group 2

$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$

Group 3

$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

$[\ x_1, \cdot, \cdot \parallel 1, 4, 7\ ]$

$[\ \cdot, y_1, \cdot \parallel 1, 4, 7\ ]$

$[\ \cdot, \cdot, z_1 \parallel 1, 4, 7\ ]$

Recall: From [GPS21]

$[\ x_1, y_1, z_1 \parallel 1, 4, 7\ ]$

Example: Network Routing

Group 1	Group 2	Group 3
$[\ x_1, x_2, x_3 \parallel 1, 2, 3\ ]$	$[\ y_1, y_2, y_3 \parallel 4, 5, 6\ ]$	$[\ z_1, z_2, z_3 \parallel 7, 8, 9\ ]$

- Suppose we want to collect secrets x_1, y_1, z_1, and store them at $(10,11,12)$

- Step 1: Locally compute $[\ x_1, y_1, z_1 \parallel 1, 4, 7\ ]$

- **Step 2: Sharing Transformation**

$$[\ x_1, y_1, z_1 \parallel 1, 4, 7\ ] \quad \rightarrow \quad [\ x_1, y_1, z_1 \parallel 10, 11, 12\ ]$$

Example: Network Routing

- Our Sharing Transformation

$$[\ {\color{red}x_1, y_1, z_1} \parallel 1,\ 4,\ 7\ ] \quad \rightarrow \quad [\ {\color{red}x_1, y_1, z_1} \parallel 10, 11, 12\ ]$$

Each Sharing Transformation is Different

Previous Techniques do not Work

Fan Out Gates

- Omitted Case: What if we want to prepare $[x_1, x_1, y_1]$

- Idea:
  1. Prepare a sharing with unrepeated secrets

$$[\ x_1,\ \cdot,\ y_1\ ]$$

  2. Sharing Transformation

$$[\ x_1,\ \cdot,\ y_1\ ] \longrightarrow [\ x_1,\ x_1,\ y_1\ ]$$

Addition Gates

- Evaluate in batches of k gates (from the same circuit layer)
- One packed Sharing for 1st input wire [x], another for 2nd input wire [y]
- Say output wire secret storing positions are 10, 11, 12. Need to collect and put secrets positions 10, 11, 12 in both [x] and [y].
- Finally, addition can be done locally [z] = [x] + [y]

Multiplication Gates

- Evaluate in batches of k gates (from the same circuit layer)
- One packed Sharing for 1st input wire [x], another for 2nd input wire [y]
- Need to collect and put the secrets in default positions (say 1, 2, 3,….)
- Use the packed Beaver Triple Technique for multiplication

Multiplication using Packed Beaver Triples

- Compute $[\mathbf{a}]_{n-k}$, $[\mathbf{b}]_{n-k}$, $[\mathbf{c}]_{n-k}$ where **a**, **b** are random vectors in F^k and $\mathbf{c} = \mathbf{a}*\mathbf{b}$
- Parties locally compute $[\mathbf{x+a}]_{n-k}$, $[\mathbf{y+b}]_{n-k}$, and send to P1
- P1 recovers **x+a**, **y+b** and distributes sharings $[\mathbf{x+a}]_{k-1}$, $[\mathbf{y+b}]_{k-1}$
- All parties can compute $[\mathbf{z}]_{n-1}$ (via a local operation)
- Final step: reduce degree from n-1 to n-k, and move secrets to correct positions

Summary

- Use packed secret sharing technique to evaluate a single circuit

  ⇒ Reduce the cost by a factor of $O(n)$ in the sub-optimal corruption setting

- Main Difficulty: Network Routing

A Simple and Efficient Approach for Large Fields via

Sparsely Packed Shamir Sharings ➕ Efficient Sharing Transformation

Our Results: Malicious Setting

Main Theorem --- Malicious (*Informal*).

For an arithmetic circuit C over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq 2^{\kappa}$, and for all constant $0 < \epsilon \leq 2/3$ and $t = (1-\epsilon) \cdot n$, there is a malicious IT MPC which computes C with $O(|\mathrm{C}|)$ elements of both preprocessing data and communication complexity.

Techniques

- Information-theoretic MAC.
- Computing $([\boldsymbol{x}], [\boldsymbol{\gamma} * \boldsymbol{x}])$ for Each Batch of Secrets $\boldsymbol{x}$.

Our Results: Small Fields

Main Theorem --- Semi-Honest (*Informal*).

Field Size Requirement

$|\mathbb{F}| \geq |\mathbb{C}| + n \quad \Rightarrow \quad |\mathbb{F}| \geq 2n$

Techniques

- Efficient Sharing Transformation Protocols.
- Hall's Marriage Theorem. [GPS21]

Thank You!