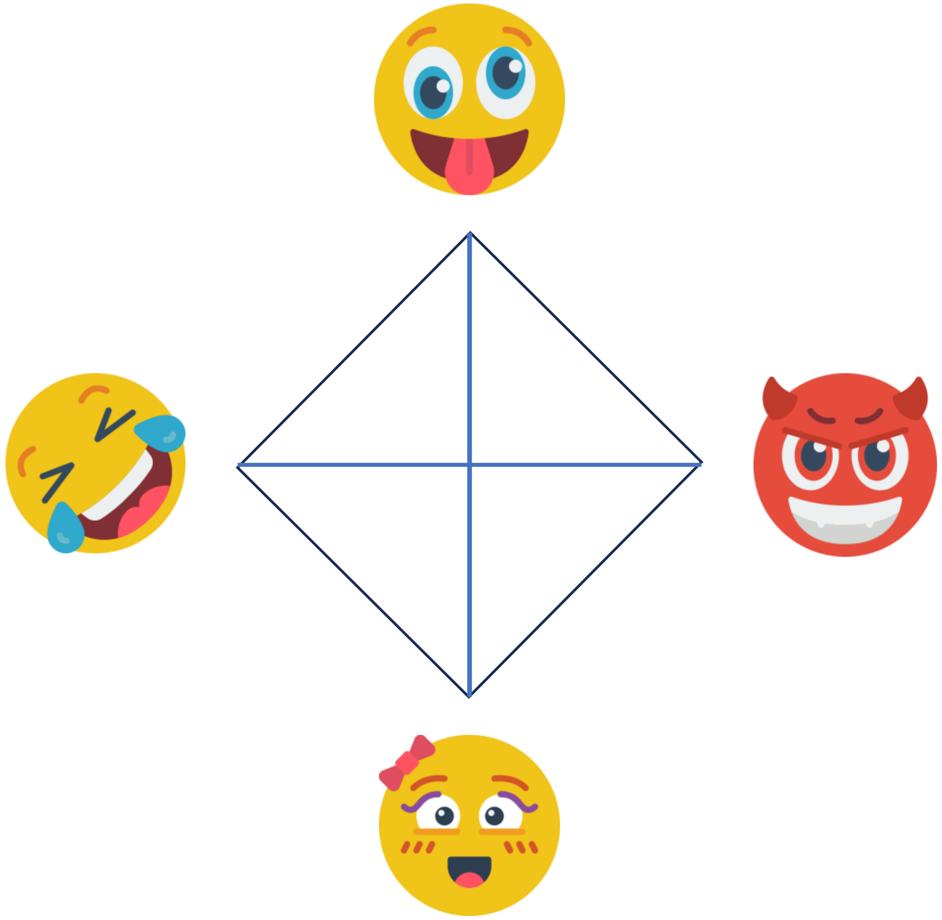
Perfectly-Secure MPC with Constant Online Communication Complexity

Yifan Song
Tsinghua University & Shanghai Qi Zhi Institute
Xiaxi Ye
Tsinghua University

Multiparty Computation



Setting

- n parties
- t corrupted parties
- Optimal resilience: $n = 3t + 1$
- Synchronous network

Goal

- Perfect security

Communication Complexity

Reference	Overall Communication	Online Communication	Security	Adversary
[BH08]	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	Optimal Resilience $n = 3t + 1$	Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$		

$|C|$: circuit size, D: circuit depth, n: number of parties, counted by field elements

Communication Complexity

Reference	Overall Communication	Online Communication	Security	Adversary
[BH08]	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	Optimal Resilience $n = 3t + 1$	Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$		
[DN07]	$O(\|C\| \cdot n)$	$O(\|C\| \cdot n)$	Optimal Resilience $n = 2t + 1$	Semi-honest
[EGPS22]	$O(\|C\| \cdot n)$	$O(\|C\|)$		

$|C|$: circuit size, D: circuit depth, n: number of parties, counted by field elements

Communication Complexity

Reference	Overall Communication	Online Communication	Security	Adversary
[BH08]	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	Optimal Resilience $n = 3t + 1$	Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$		
[DN07]	$O(\|C\| \cdot n)$	$O(\|C\| \cdot n)$	Optimal Resilience $n = 2t + 1$	Semi-honest
[EGPS22]	$O(\|C\| \cdot n)$	$O(\|C\|)$		

$|C|$: circuit size, D: circuit depth, n: number of parties, counted by field elements

Is it possible to construct a perfectly secure MPC protocol with GOD

such that the online communication complexity per gate is $O(1)$

while the overall communication remains $O(n)$?

Why Constant Online Communication?

- Online efficiency is important as the preprocessing phase which only depends on the circuit size can be done in the idle time.

- Amortized online communication complexity per party decreases as the increase of the number of parties!

Our Result

Reference	Overall Communication	Online Communication	Security	Adversary
[BH08]	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	$O(\|C\| \cdot n + D \cdot n^2 + n^3)$	Optimal Resilience $n = 3t + 1$	Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$		
Our result	$O(\|C\| \cdot n + D \cdot n^2 + n^5)$	$O(\|C\| + D \cdot n + n^5)$		
[DN07]	$O(\|C\| \cdot n)$	$O(\|C\| \cdot n)$	Optimal Resilience $n = 2t + 1$	Semi-honest
[EGPS22]	$O(\|C\| \cdot n)$	$O(\|C\|)$		

Theorem.

Let $n = 3t + 1$. For any arithmetic circuit C over $\mathbb{F}$ of size $|\mathbb{F}| \geq 2n$ of size $|C|$ and depth D, there is an information-theoretic MPC protocol against a fully malicious adversary controlling at most t corrupted parties with perfect security. The communication is $O(|C| + D \cdot n + n^5)$ elements for the online phase and $O(|C| \cdot n + D \cdot n^2 + n^4)$ elements for the offline phase.

Limitations of Our Result

Packed Shamir secret sharing, hyper-invertible matrix

Limitation 1: Only work for finite fields of size larger than $2n$

Dispute control framework

Limitation 2: Round complexity grows with number of parties

A Relative Mention – Round complexity

- A line of works [ALR11, AAY22, AAPP23] focuses on optimizing communication without $O(n)$ overhead in the round complexity.

Reference	Overall Communication	Online Communication	Round complexity	Security
[AAPP23]	$O(\lvert C\rvert \cdot n + D \cdot n^2 + n^4)$		$O(D)$	$n = 3t + 1$ Malicious with GOD
[GLS19]	$O(\lvert C\rvert \cdot n + n^3)$	$O(\lvert C\rvert \cdot n + n^3)$	$O(D + n)$	
Our result	$O(\lvert C\rvert \cdot n + D \cdot n^2 + n^5 \cdot \log n)$	$O(\lvert C\rvert + D \cdot n + n^5 \cdot \log n)$	$O(D + n^2)$	

If expected constant-round BA and BC in [AC24] are used.

A Relative Mention – Round complexity

- A line of works [ALR11, AAY22, AAPP23] focuses on optimizing communication without $O(n)$ overhead in the round complexity.

Reference	Overall Communication	Online Communication	Round complexity	Security
[AAPP23]	$O(\vert C\vert \cdot n + D \cdot n^2 + n^4)$		$O(D)$	$n = 3t + 1$ Malicious with GOD
[GLS19]	$O(\vert C\vert \cdot n + n^3)$	$O(\vert C\vert \cdot n + n^3)$	$O(D + n)$	
Our result	$O(\vert C\vert \cdot n + D \cdot n^2 + n^5 \cdot \log n)$	$O(\vert C\vert + D \cdot n + n^5 \cdot \log n)$	$O(D + n^2)$	

If expected constant-round BA and BC in [AC24] are used.

A Relative Mention – Circuit depth overhead

- [GLS19] removes the quadratic communication overhead in the circuit depth.

Reference	Overall Communication	Online Communication	Round complexity	Security
[AAPP23]	$O(\|C\| \cdot n + D \cdot n^2 + n^4)$		$O(D)$	$n = 3t + 1$ Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$	$O(D + n)$	
Our result	$O(\|C\| \cdot n + D \cdot n^2 + n^5 \cdot \log n)$	$O(\|C\| + D \cdot n + n^5 \cdot \log n)$	$O(D + n^2)$	

A Relative Mention – Circuit depth overhead

- [GLS19] removes the quadratic communication overhead in the circuit depth.

Reference	Overall Communication	Online Communication	Round complexity	Security
[AAPP23]	$O(\lvert C\rvert \cdot n + D \cdot n^2 + n^4)$		$O(D)$	$n = 3t + 1$ Malicious with GOD
[GLS19]	$O(\lvert C\rvert \cdot n + n^3)$	$O(\lvert C\rvert \cdot n + n^3)$	$O(D + n)$	
Our result	$O(\lvert C\rvert \cdot n + D \cdot n^2 + n^5 \cdot \log n)$	$O(\lvert C\rvert + D \cdot n + n^5 \cdot \log n)$	$O(D + n^2)$	

A Relative Mention – Circuit depth overhead

- [GLS19] removes the quadratic communication overhead in the circuit depth.

Reference	Overall Communication	Online Communication	Round complexity	Security
[AAPP23]	$O(\|C\| \cdot n + D \cdot n^2 + n^4)$		$O(D)$	$n = 3t + 1$ Malicious with GOD
[GLS19]	$O(\|C\| \cdot n + n^3)$	$O(\|C\| \cdot n + n^3)$	$O(D + n)$	
Our result	$O(\|C\| \cdot n + D \cdot n^2 + n^5 \cdot \log n)$	$O(\|C\| + D \cdot n + n^5 \cdot \log n)$	$O(D + n^2)$	

Can we achieve the best among three works?

Outline

- **Review: semi-honest protocol in [EGPS22]**

- Towards full security via dispute control:

  verification + identifying dispute pairs

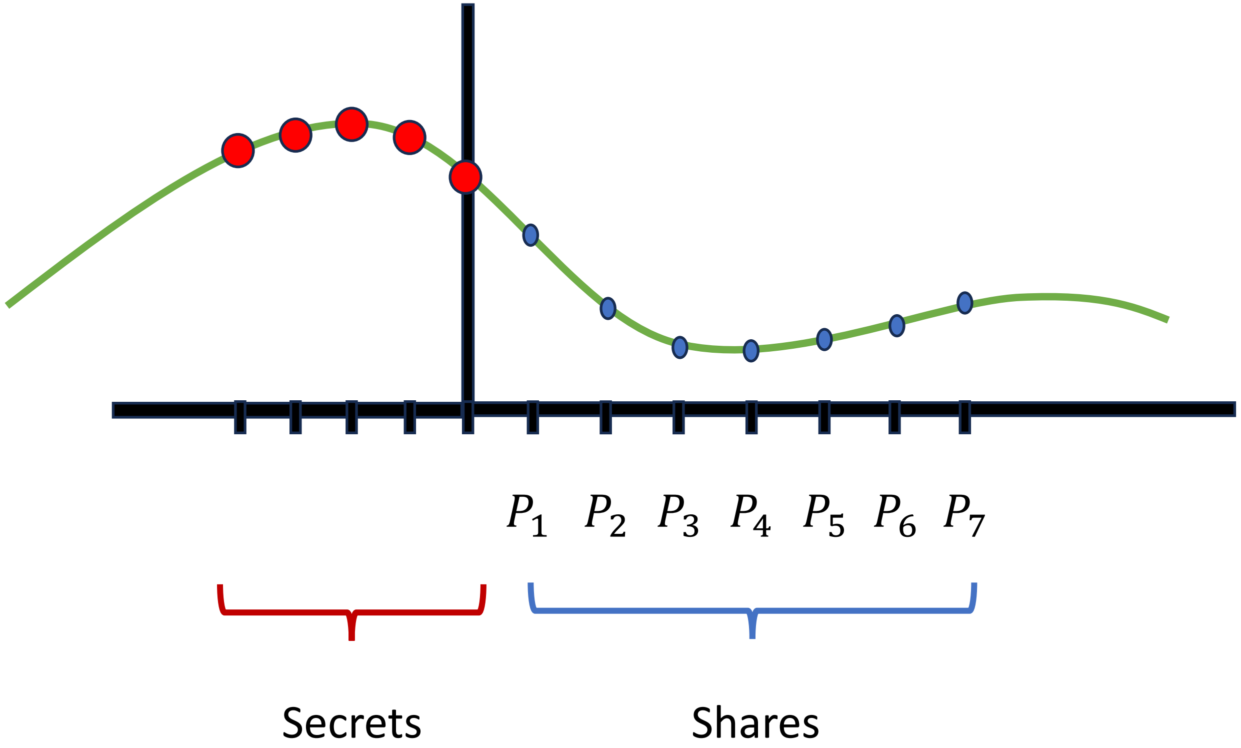
- Towards general circuits via sharing transformation

Packed Shamir Secret Sharing

Secrets: $\boldsymbol{s} = (s_1, s_2, \dots, s_k)$

Parameters:

- pack size k
- degree-$(t + k - 1)$



Use a degree-$(t + k - 1)$ polynomial:

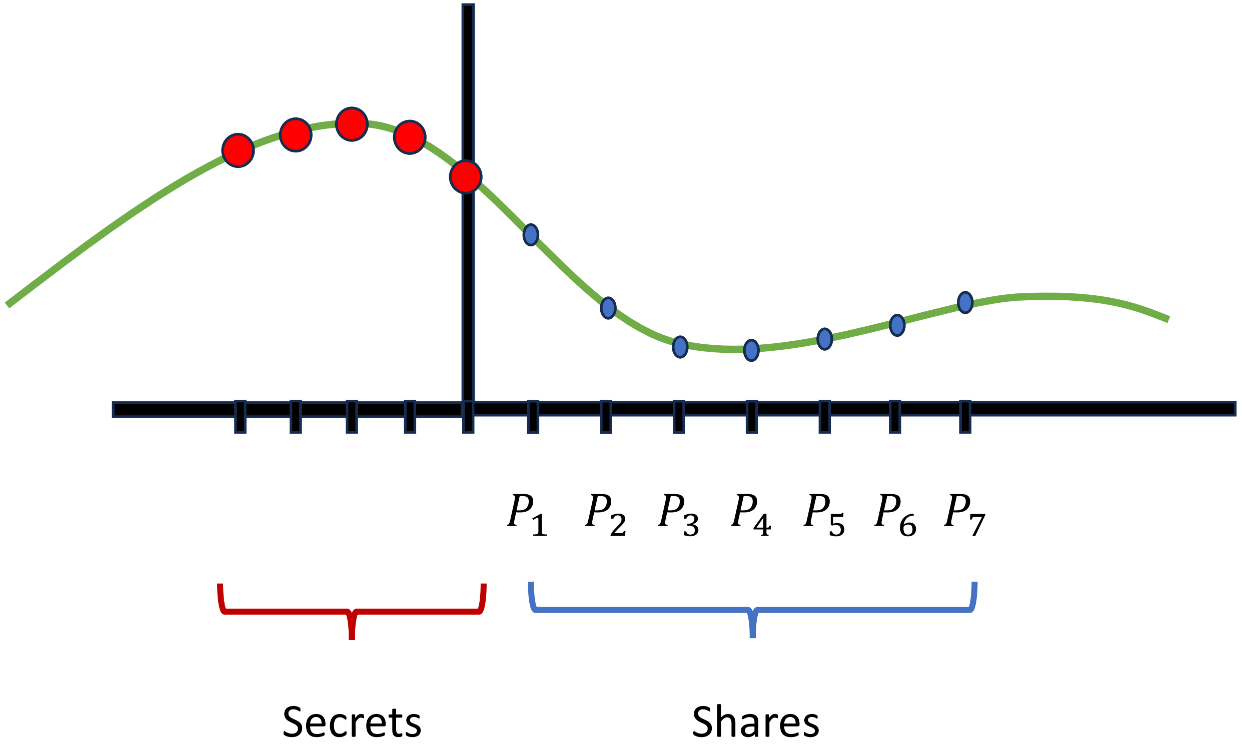
- Each share is an evaluation point of this polynomial.
- Any t shares are independent of the secrets.
- Any $t + k$ shares can reconstruct the secrets.

Packed Shamir Secret Sharing

Secrets: $\boldsymbol{s} = (s_1, s_2, \ldots, s_k)$

Parameters:

- pack size k
- degree-$(t + k - 1)$



Use a degree-$(t + k - 1)$ polynomial:

- Linearly homomorphic.

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

- Multiplicative friendly.

$$[\boldsymbol{c}]_{k-1} \cdot [\boldsymbol{x}]_{t+k-1} = [\boldsymbol{c} * \boldsymbol{x}]_{t+2k-2}$$

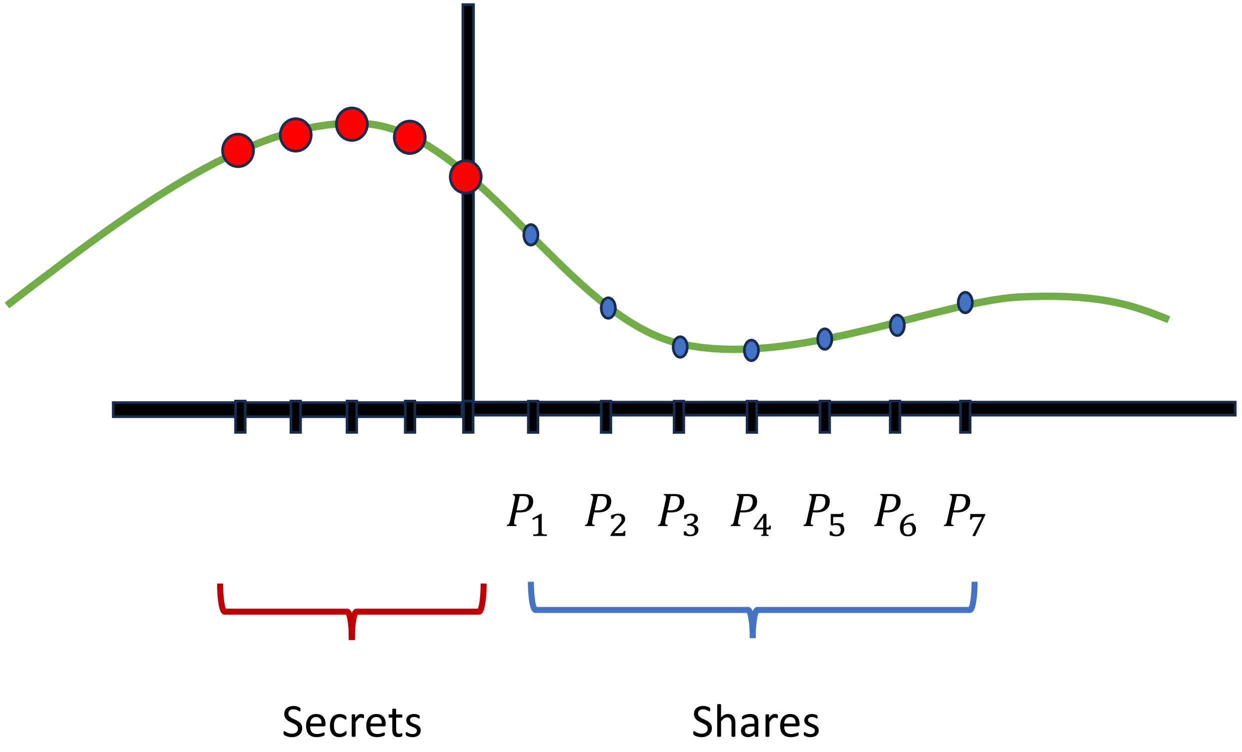
Packed Shamir Secret Sharing

$k = (t+2)/2$

Secrets: $\boldsymbol{s} = (s_1, s_2, \dots, s_k)$

Parameters:

- pack size k
- degree-$(t+k-1)$



Use a degree-$(t+k-1)$ polynomial:

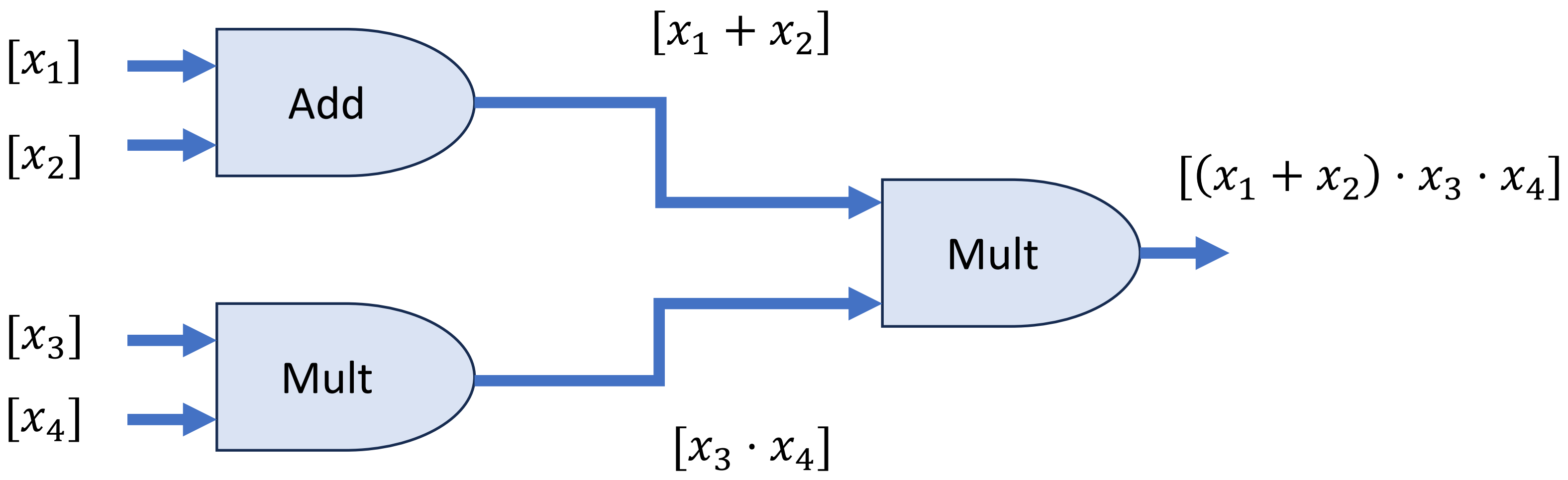
- Linearly homomorphic.

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

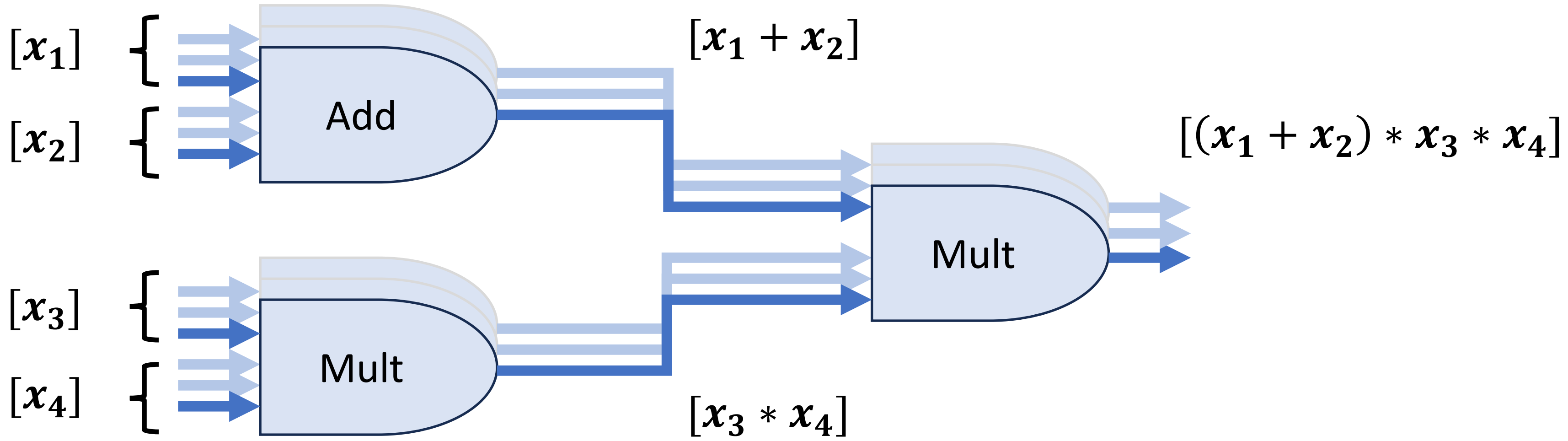
- Multiplicative friendly.

$$[\boldsymbol{c}]_{k-1} \cdot [\boldsymbol{x}]_{t+k-1} = [\boldsymbol{c} * \boldsymbol{x}]_{t+2k-2}$$

Generic Approach

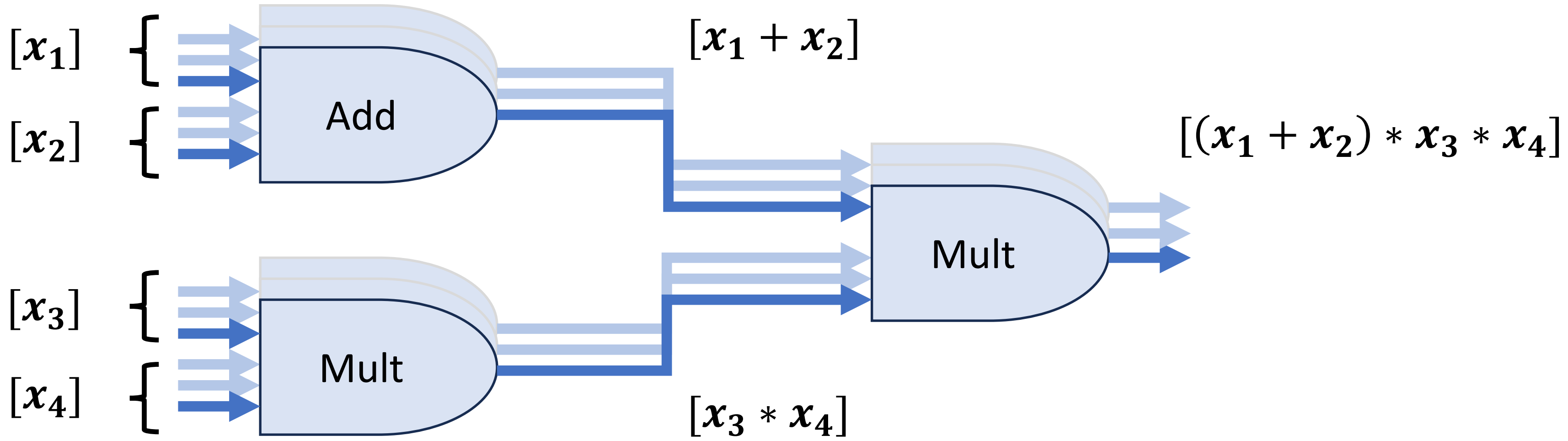


Generic Approach (SIMD Circuit)



Generic Approach (SIMD Circuit)

$k = \Omega(n)$ for communication benefits.



Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} \longleftarrow \text{1}$$

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

Reconstruct $\boldsymbol{x} + \boldsymbol{a}$, $\boldsymbol{y} + \boldsymbol{b}$

$[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y} + \boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

Reconstruct $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

Compute $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$

$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$

1 $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ ←

Reconstruct $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$

Compute $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

2 $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ →

Multiplication Protocol adapted from [EGPS22]

Multiplication

- Preprocessing: $([\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1})$
- Input: $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}$.

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

1 $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

Reconstruct $\boldsymbol{x}+\boldsymbol{a}$, $\boldsymbol{y}+\boldsymbol{b}$

Compute $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

2 $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

$$[\boldsymbol{z}]_{t+2k-2} = [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_{t+k-1}$$

$$-[\boldsymbol{y}+\boldsymbol{b}]_{k-1} \cdot [\boldsymbol{a}]_{t+k-1} + [\boldsymbol{c}]_{t+k-1}$$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

$$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Reconstruct $\boldsymbol{z} + \boldsymbol{r}$

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2}$

3

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Reconstruct $\boldsymbol{z}+\boldsymbol{r}$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Compute $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Reconstruct $\boldsymbol{z} + \boldsymbol{r}$

3 $[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2}$

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Compute $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$

4 $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Reconstruct $\boldsymbol{z} + \boldsymbol{r}$

3 $[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2}$

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Compute $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$

4 $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$

$[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z} + \boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{t+k-1}$

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

Reconstruct $\boldsymbol{z}+\boldsymbol{r}$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

Compute $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

$[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

4

$[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{t+k-1}$

If $\mathrm{k} = \Omega(n)$, $O(1)$ elements per gate.

Multiplication Protocol adapted from [EGPS22]

Degree Reduction

- Preprocessing: $([\boldsymbol{r}]_{t+2k-2}, [\boldsymbol{r}]_{t+k-1})$.
- Output: $[\boldsymbol{x} * \boldsymbol{y}]_{t+k-1}$

3

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

Reconstruct $\boldsymbol{z}+\boldsymbol{r}$

4

Compute $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

$[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

$[\boldsymbol{z}]_{t+k-1} = [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_{t+k-1}$

If $\mathrm{k} = \Omega(n)$, $O(1)$ elements per gate.

Switch to **malicious** setting?

Multiplication Protocol adapted from [EGPS22]

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

2

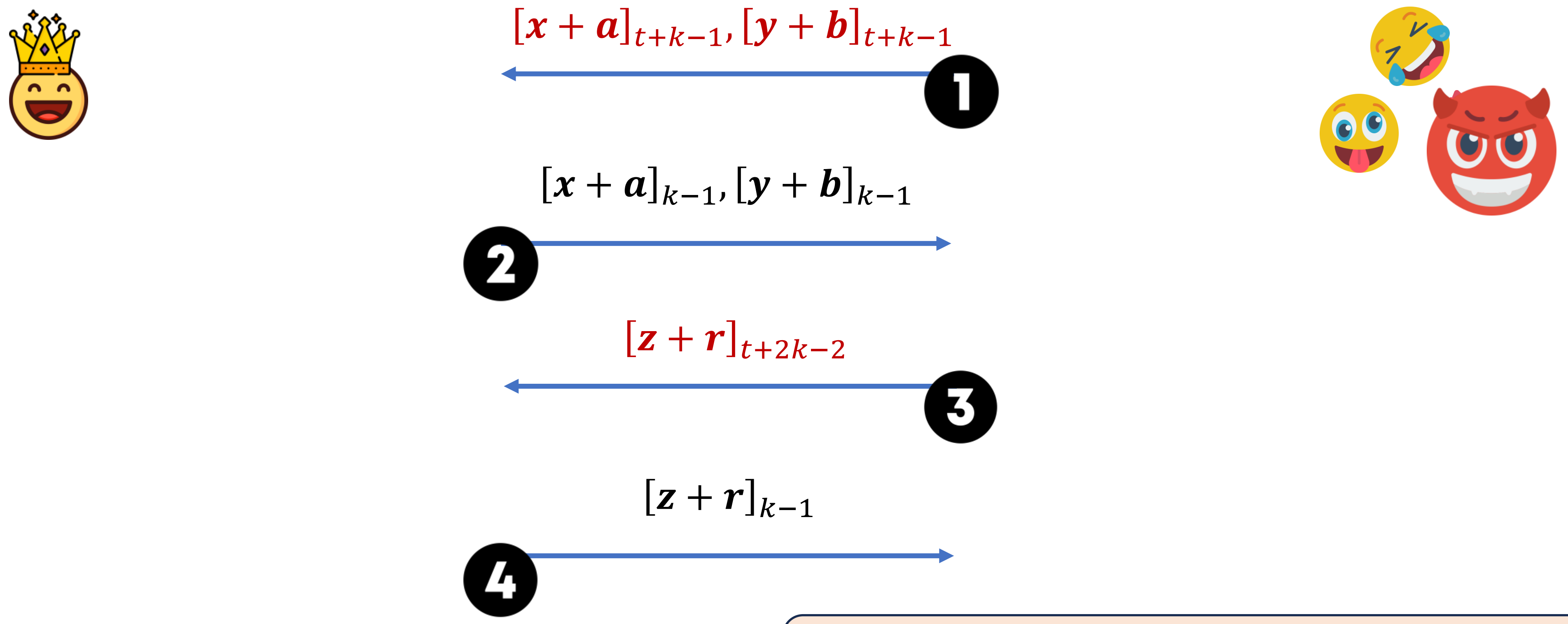
$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

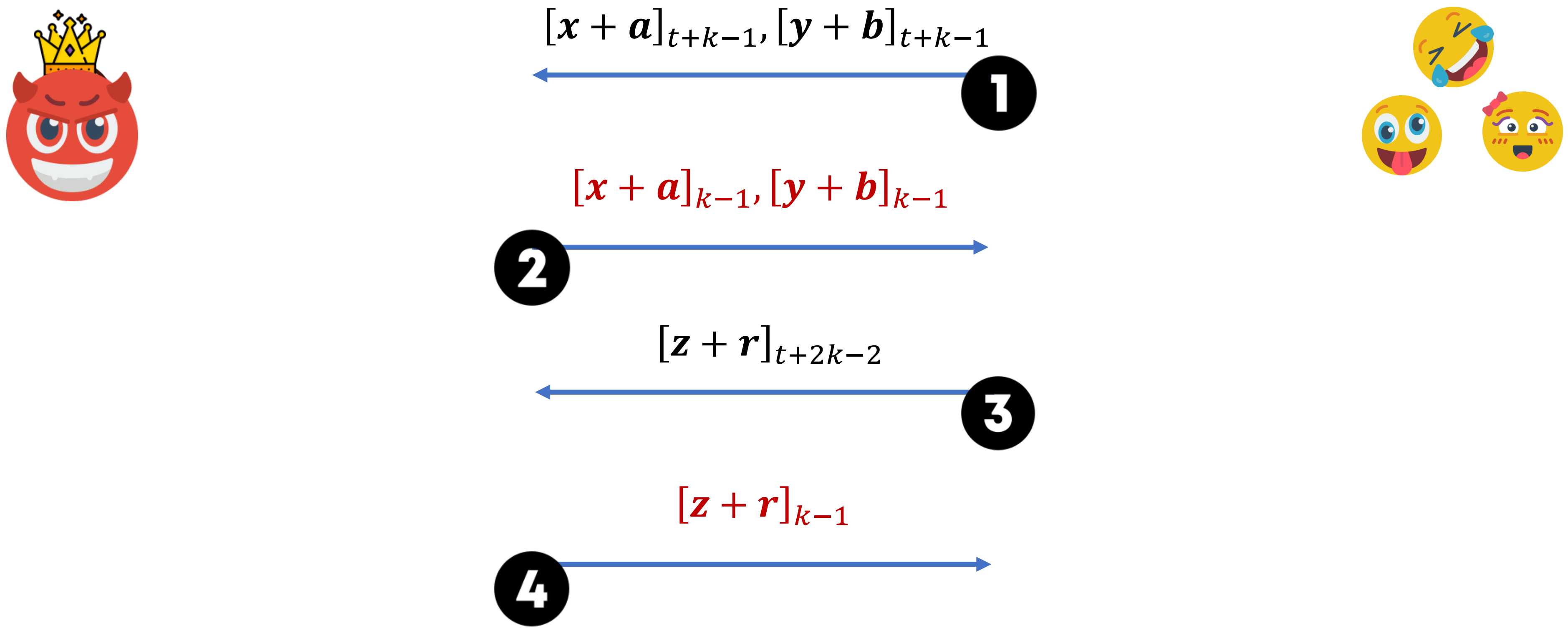
4

Multiplication Protocol adapted from [EGPS22]



Adversary may send incorrect shares to P_{king}.

Multiplication Protocol adapted from [EGPS22]



Multiplication Protocol adapted from [EGPS22]

1. $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

2. $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

3. $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

4. $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

Adversary may distribute sharings NOT of degree $k-1$ or with incorrect secrets.

Outline

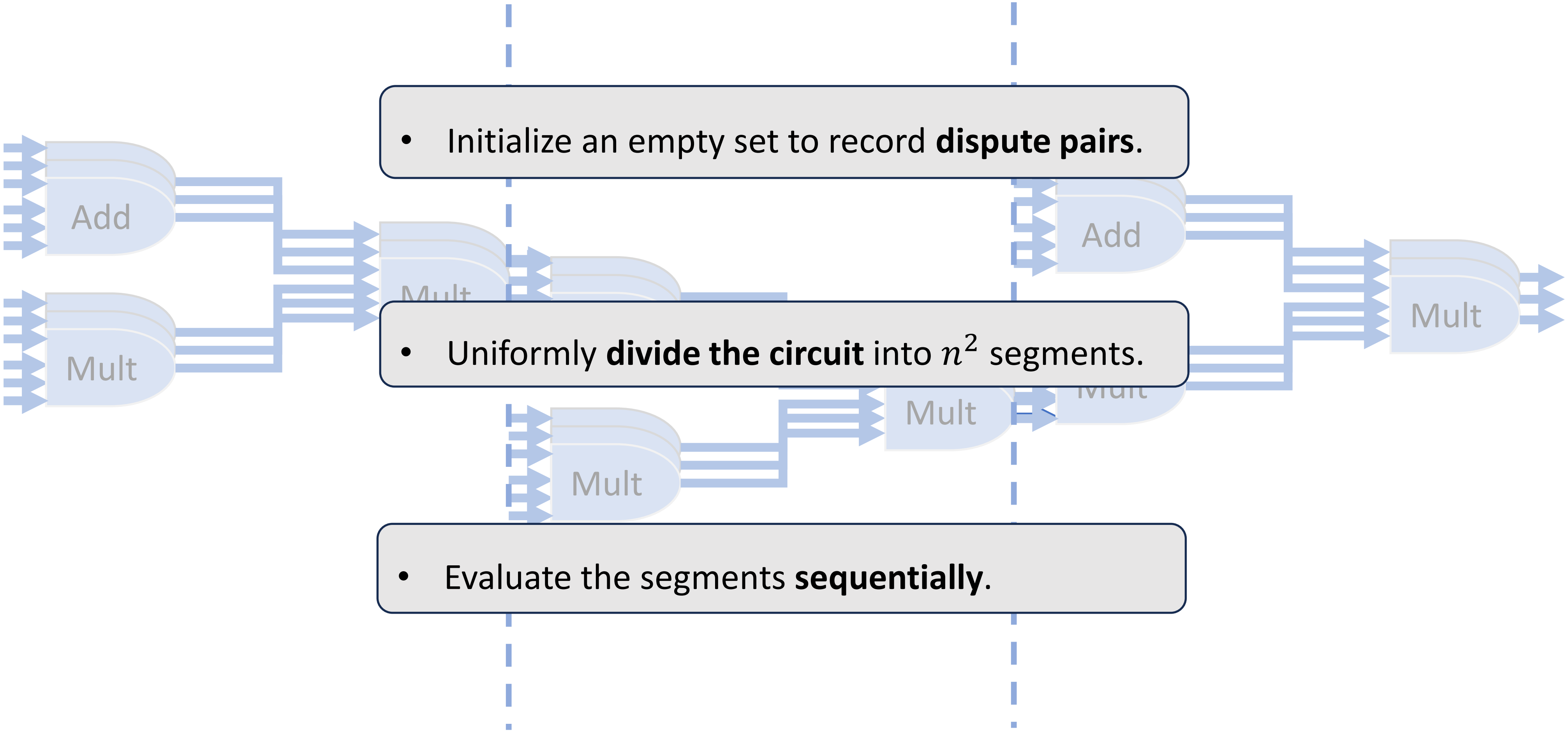
- Review: semi-honest protocol in [EGPS22]

- **<u>Towards full security via dispute control</u>**:

verification + identifying dispute pairs

- Towards general circuits via sharing transformation

Towards GOD: Dispute Control Framework [BH06]



- Initialize an empty set to record **dispute pairs**.
- Uniformly **divide the circuit** into n^2 segments.
- Evaluate the segments **sequentially**.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

- Evaluate the segment.
- Verify the computation.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

- Evaluate the segment.
- Verify the computation.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

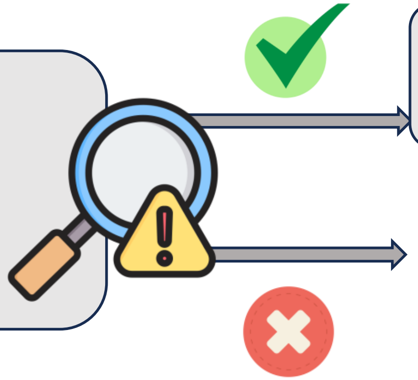
- Evaluate the segment.
- Verify the computation.

Evaluate the next segment.

Towards GOD: Dispute Control Framework [BH06]

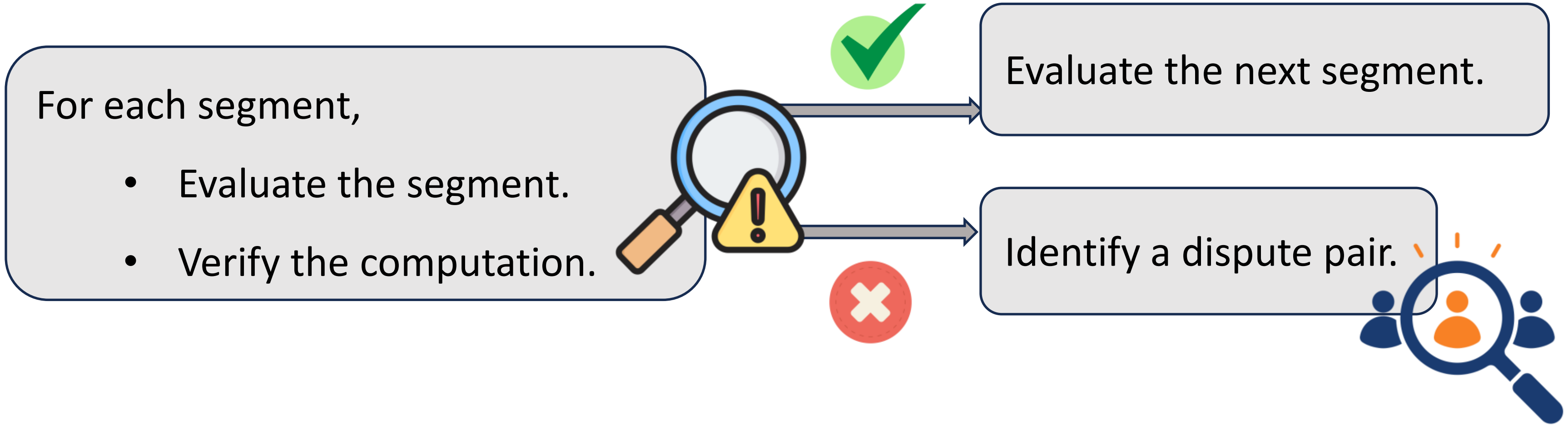
For each segment,

- Evaluate the segment.
- Verify the computation.

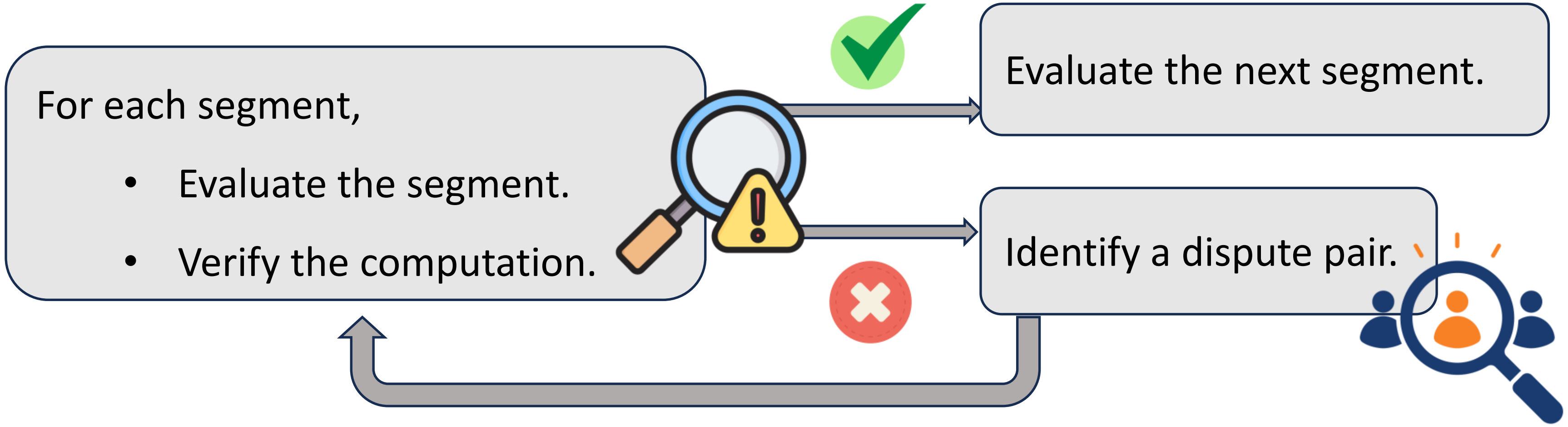


Evaluate the next segment.

Towards GOD: Dispute Control Framework [BH06]



Towards GOD: Dispute Control Framework [BH06]



Towards GOD: Dispute Control Framework [BH06]

For each segment,

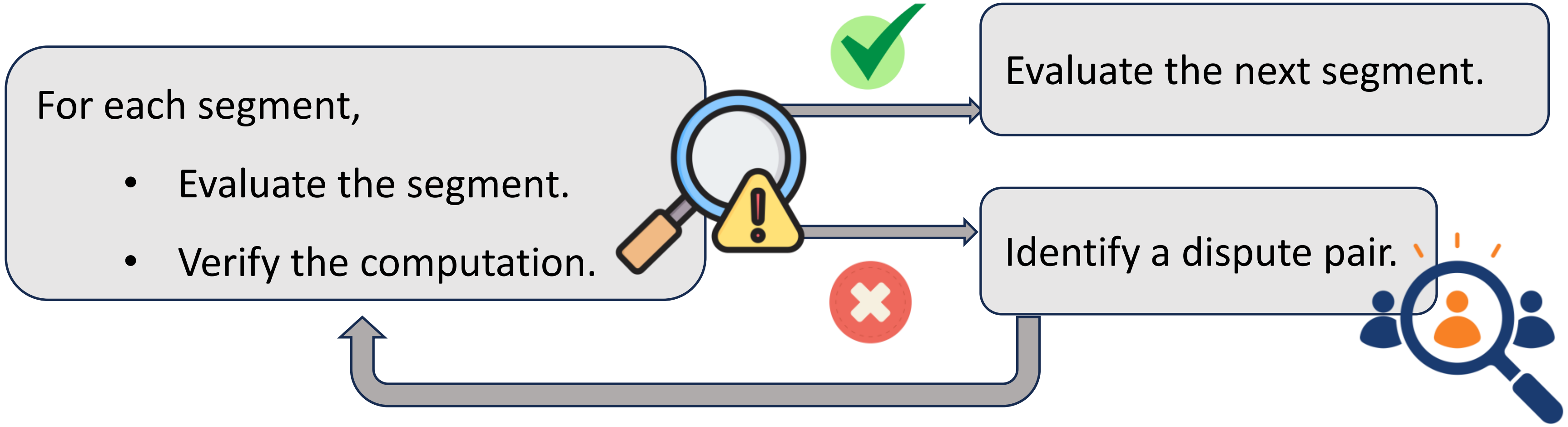
- Evaluate the segment.
- Verify the computation.

Evaluate the next segment.

Identify a dispute pair.

Corrupted parties will be eliminated. Find a **relay** for each dispute pair [BFO12].

Towards GOD: Dispute Control Framework [BH06]



Corrupted parties will be eliminated. Find a **relay** for each dispute pair [BFO12].

→

Two disputed parties will **never** talk to each other.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

- Evaluate the segment.
- Verify the computation.

Evaluate the next segment.

Identify a dispute pair.

Corrupted parties will be eliminated. Find a **relay** for each dispute pair [BFO12].

Two disputed parties will **never** talk to each other.

Always find a **new** dispute pair.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

- Evaluate the segment.
- Verify the computation.

Evaluate the next segment.

Identify a dispute pair.

Corrupted parties will be eliminated. Find a **relay** for each dispute pair [BFO12].

Two disputed parties will **never** talk to each other.

Always find a **new** dispute pair.

Re-evaluation occurs at most $O(n^2)$ times.

Towards GOD: Dispute Control Framework [BH06]

For each segment,

- Evaluate the segment.
- Verify the computation.

Evaluate the next segment.

Identify a dispute pair.

Corrupted parties will be eliminated. Find a **relay** for each dispute pair [BFO12].

Two disputed parties will **never** talk to each other.

Always find a **new** dispute pair.

Re-evaluation occurs at most $O(n^2)$ times.

Communication at most doubles.

Outline

- Review: semi-honest protocol in [EGPS22]

- **<u>Towards full security via dispute control</u>**:

  <u>verification</u> + identifying dispute pairs

- Towards general circuits via sharing transformation

Towards GOD: Verification - 1

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$$

Adversary may send incorrect shares to P_{king}.

Towards GOD: Verification - 1

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$$

Adversary may send incorrect shares to P_{king}.

The whole sharing is determined by shares of honest parties.

Towards GOD: Verification - 1

$[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y} + \boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2}$

3

$$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$$

Adversary may send incorrect shares to P_{king}.

Set $(t + 2k - 2) + 1 \leq n - t$.

In particular, $k = (t + 2)/2$.

The whole sharing is determined by shares of honest parties.

Towards GOD: Verification - 1

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1}$$

$$[\boldsymbol{y}+\boldsymbol{b}]_{t+k-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1}$$

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

$$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2}$$

Set $(t+2k-2)+1 \leq n-t$.

In particular, $k=(t+2)/2$.

Adversary may send incorrect shares to P_{king}.

The whole sharing is determined by shares of honest parties.

P_{king} can detect the errors by checking whether the received shares form a valid sharing of correct degree.

Towards GOD: Verification - 2

1 $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

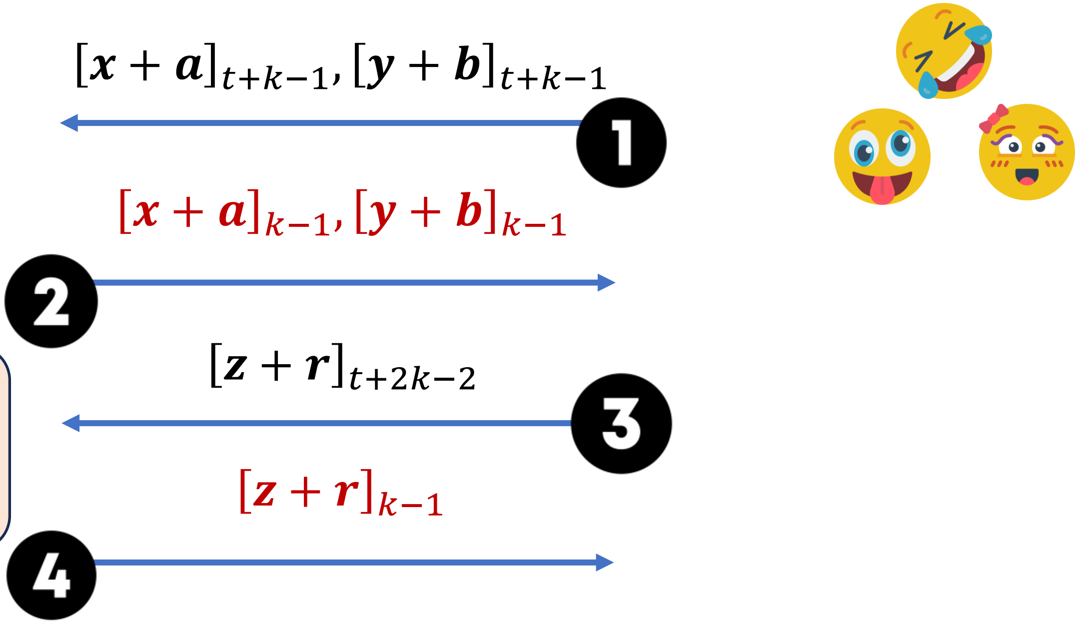
2 $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$

3 $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

4 $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$

Adversary may distribute sharings of NOT degree $k-1$ or with incorrect secrets.

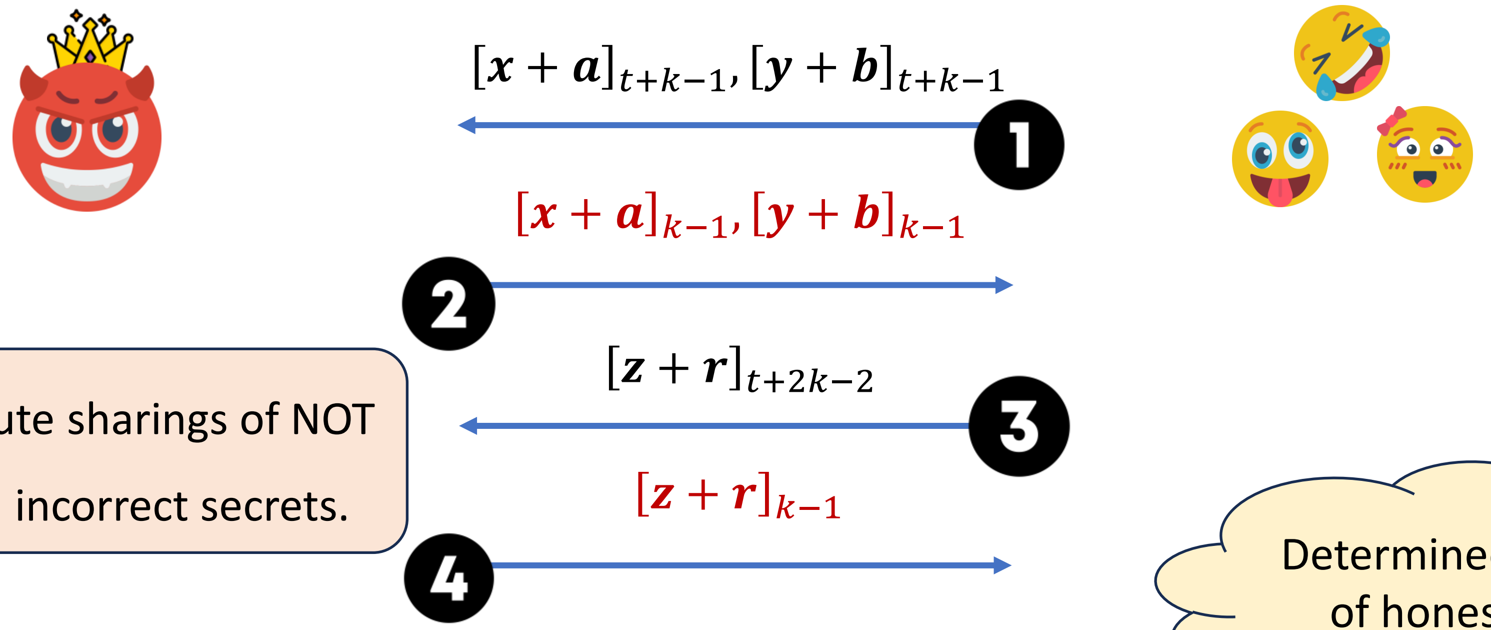
Towards GOD: Verification - 2



Adversary may distribute sharings of NOT degree $k-1$ or with incorrect secrets.

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

Towards GOD: Verification - 2



Adversary may distribute sharings of NOT degree $k-1$ or with incorrect secrets.

Determined by shares of honest parties

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$$\left.\boxed{\begin{array}{l} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{array}}\right\} \begin{array}{l} (n-t) \\ \text{pairs to be} \\ \text{checked} \end{array}$$

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$$\begin{bmatrix} [\boldsymbol{v}_1]_{t+2k-2}, [\boldsymbol{v}_1]_{k-1} \\ [\boldsymbol{v}_2]_{t+2k-2}, [\boldsymbol{v}_2]_{k-1} \\ [\boldsymbol{v}_3]_{t+2k-2}, [\boldsymbol{v}_3]_{k-1} \\ [\boldsymbol{v}_4]_{t+2k-2}, [\boldsymbol{v}_4]_{k-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} \end{bmatrix} \bullet \left.\begin{bmatrix} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{bmatrix}\right\} (n-t) \text{ pairs to be checked}$$

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$(n-t)$ pairs are checked by honest parties

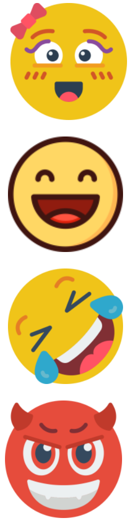
$$\begin{bmatrix} [\boldsymbol{v}_1]_{t+2k-2}, [\boldsymbol{v}_1]_{k-1} \\ [\boldsymbol{v}_2]_{t+2k-2}, [\boldsymbol{v}_2]_{k-1} \\ [\boldsymbol{v}_3]_{t+2k-2}, [\boldsymbol{v}_3]_{k-1} \\ [\boldsymbol{v}_4]_{t+2k-2}, [\boldsymbol{v}_4]_{k-1} \end{bmatrix} = \boldsymbol{M} \bullet \begin{bmatrix} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{bmatrix}$$

$(n-t)$ pairs to be checked

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$(n-t)$ pairs are checked by honest parties



$$\begin{bmatrix} [\boldsymbol{v}_1]_{t+2k-2}, [\boldsymbol{v}_1]_{k-1} \\ [\boldsymbol{v}_2]_{t+2k-2}, [\boldsymbol{v}_2]_{k-1} \\ [\boldsymbol{v}_3]_{t+2k-2}, [\boldsymbol{v}_3]_{k-1} \\ [\boldsymbol{v}_4]_{t+2k-2}, [\boldsymbol{v}_4]_{k-1} \end{bmatrix} = \boldsymbol{M} \cdot \underbrace{\begin{bmatrix} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{bmatrix}}_{(n-t)\text{ pairs to be checked}}$$

All $(n-t)$ honest parties are happy.

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$(n-t)$ pairs are checked by honest parties

$$\begin{bmatrix} [\boldsymbol{v}_1]_{t+2k-2}, [\boldsymbol{v}_1]_{k-1} \\ [\boldsymbol{v}_2]_{t+2k-2}, [\boldsymbol{v}_2]_{k-1} \\ [\boldsymbol{v}_3]_{t+2k-2}, [\boldsymbol{v}_3]_{k-1} \\ [\boldsymbol{v}_4]_{t+2k-2}, [\boldsymbol{v}_4]_{k-1} \end{bmatrix} = \boldsymbol{M} \bullet \begin{bmatrix} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{bmatrix}$$

$(n-t)$ pairs to be checked

All $(n-t)$ honest parties are happy.

Super-invertibility

All $(n-t)$ pairs are correct.

Towards GOD: Verification – 2 [BH08]

All parties check their shares of ($[\boldsymbol{u}]_{t+2k-2}, [\boldsymbol{u}]_{k-1}$).

$(n-t)$ pairs are checked by honest parties

$$\begin{bmatrix} [\boldsymbol{v}_1]_{t+2k-2}, [\boldsymbol{v}_1]_{k-1} \\ [\boldsymbol{v}_2]_{t+2k-2}, [\boldsymbol{v}_2]_{k-1} \\ [\boldsymbol{v}_3]_{t+2k-2}, [\boldsymbol{v}_3]_{k-1} \\ [\boldsymbol{v}_4]_{t+2k-2}, [\boldsymbol{v}_4]_{k-1} \end{bmatrix} = \boldsymbol{M} \bullet \begin{bmatrix} [\boldsymbol{u}_1]_{t+2k-2}, [\boldsymbol{u}_1]_{k-1} \\ [\boldsymbol{u}_2]_{t+2k-2}, [\boldsymbol{u}_2]_{k-1} \\ [\boldsymbol{u}_3]_{t+2k-2}, [\boldsymbol{u}_3]_{k-1} \end{bmatrix}$$

$(n-t)$ pairs to be checked

Batch-wise verification: $O(n)$ elements per pair.

Outline

- Review: semi-honest protocol in [EGPS22]

- **Towards full security via dispute control**:

  verification + **identifying dispute pairs**

- Towards general circuits via sharing transformation

Towards GOD: Identifying Dispute Pairs

Verification fails

Towards GOD: Identifying Dispute Pairs

Verification fails

Verification – 1 fails

P_{king} complains the received sharings are incorrect.

$[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$

1

$[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$

3

Towards GOD: Identifying Dispute Pairs

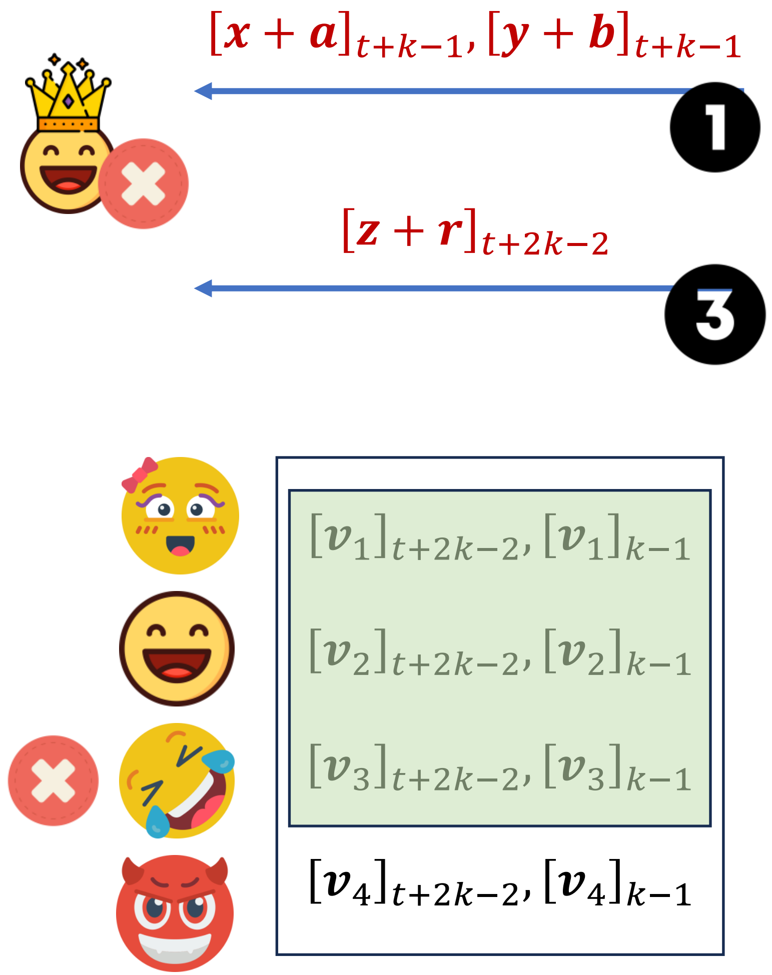
Verification fails

Verification – 1 fails

P_{king} complains the received sharings are incorrect.

Verification – 2 fails

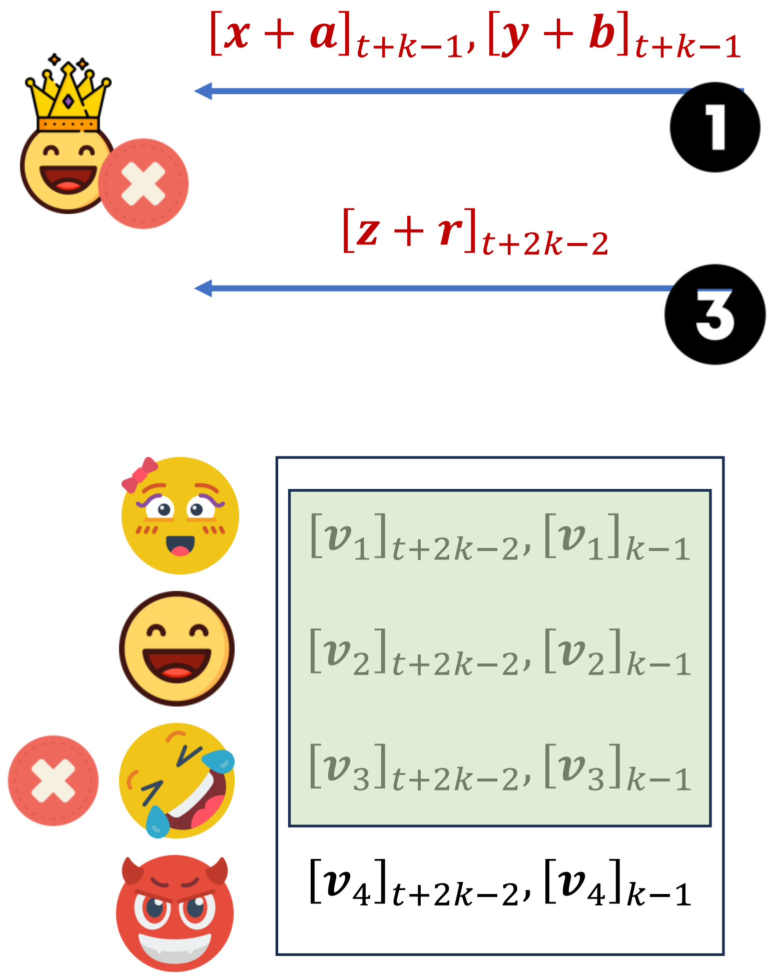
P_i complains the received $([\boldsymbol{v}_i]_{t+2k-2}, [\boldsymbol{v}_i]_{k-1})$ is incorrect.



Towards GOD: Identifying Dispute Pairs

Verification fails

- Verification – 1 fails → P_{king} complains the received sharings are incorrect.
- Verification – 2 fails → P_i complains the received $([\boldsymbol{v}_i]_{t+2k-2}, [\boldsymbol{v}_i]_{k-1})$ is incorrect. → Ask P_{king} to provide a correct version.



Towards GOD: Identifying Dispute Pairs

Verification fails

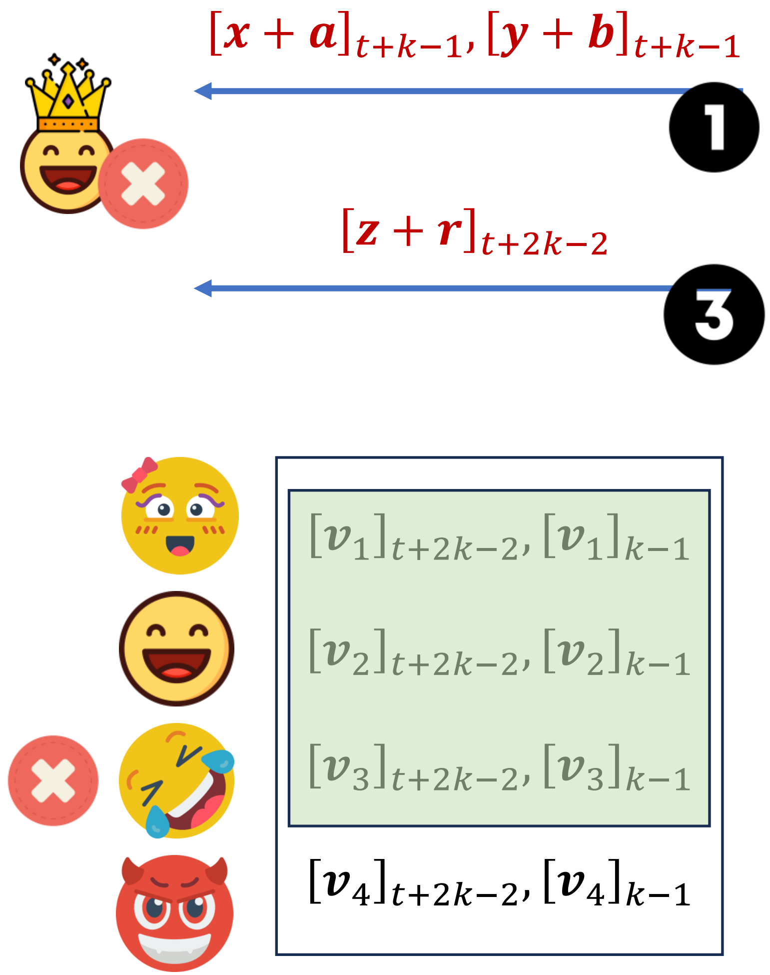
Verification – 1 fails → P_{king} complains the received sharings are incorrect.

Verification – 2 fails → P_i complains the received $([\boldsymbol{v}_i]_{t+2k-2}, [\boldsymbol{v}_i]_{k-1})$ is incorrect.

↓

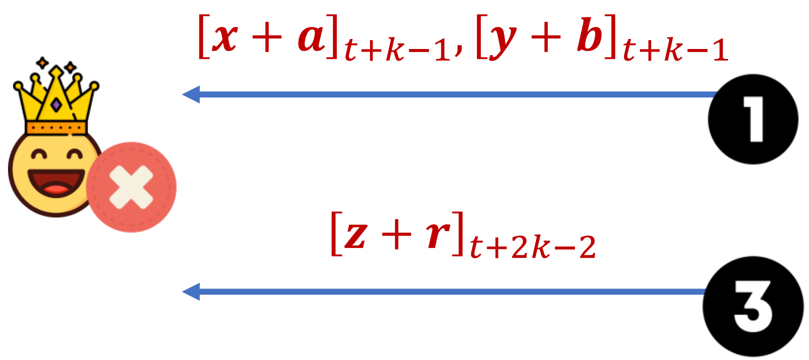
Ask P_{king} to provide a correct version.

→ A dispute pair



Towards GOD: Identifying Dispute Pairs - 1

Verification – 1 fails → P_{king} complains the received sharings are inconsistent.

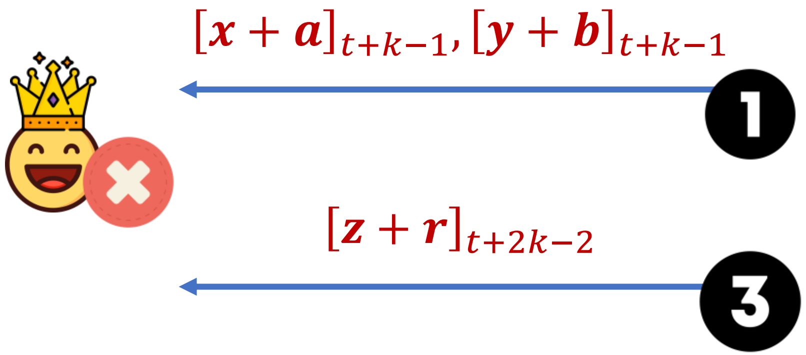


Towards GOD: Identifying Dispute Pairs - 1

Verification – 1 fails

P_{king} complains the received sharings are inconsistent.

P_{king} cannot identify incorrect shares



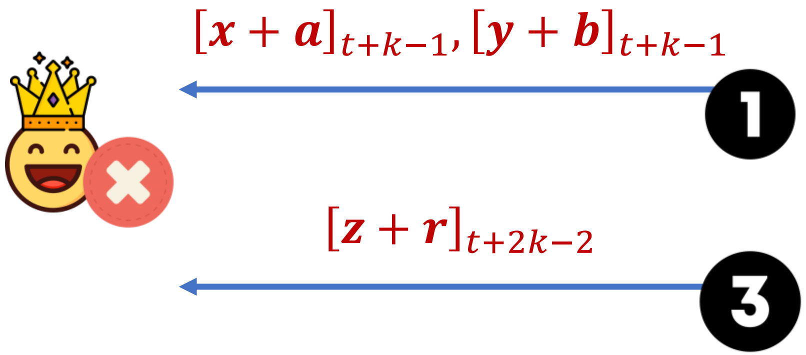
Towards GOD: Identifying Dispute Pairs - 1

Verification – 1 fails

P_{king} complains the received sharings are inconsistent.

P_{king} cannot identify incorrect shares

Resort to deg-t sharings



Towards GOD: Identifying Dispute Pairs - 1

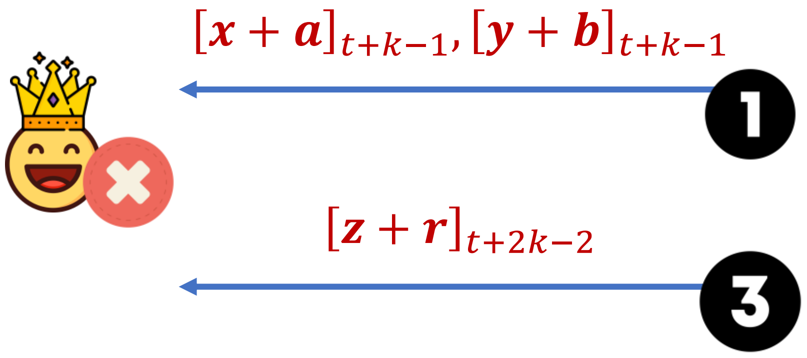
Verification – 1 fails

P_{king} complains the received sharings are inconsistent.

P_{king} cannot identify incorrect shares

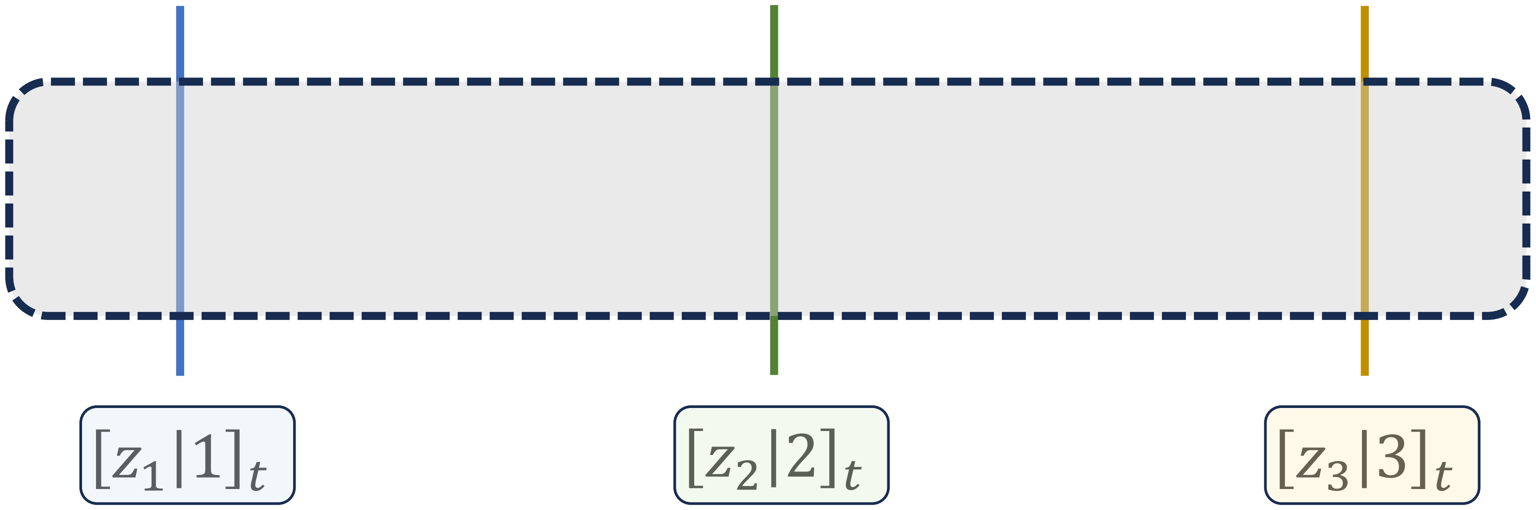
Resort to deg-t sharings

Need to compute deg-t sharings with $O(1)$ elements per gate.



Towards GOD: Identifying Dispute Pairs - 1

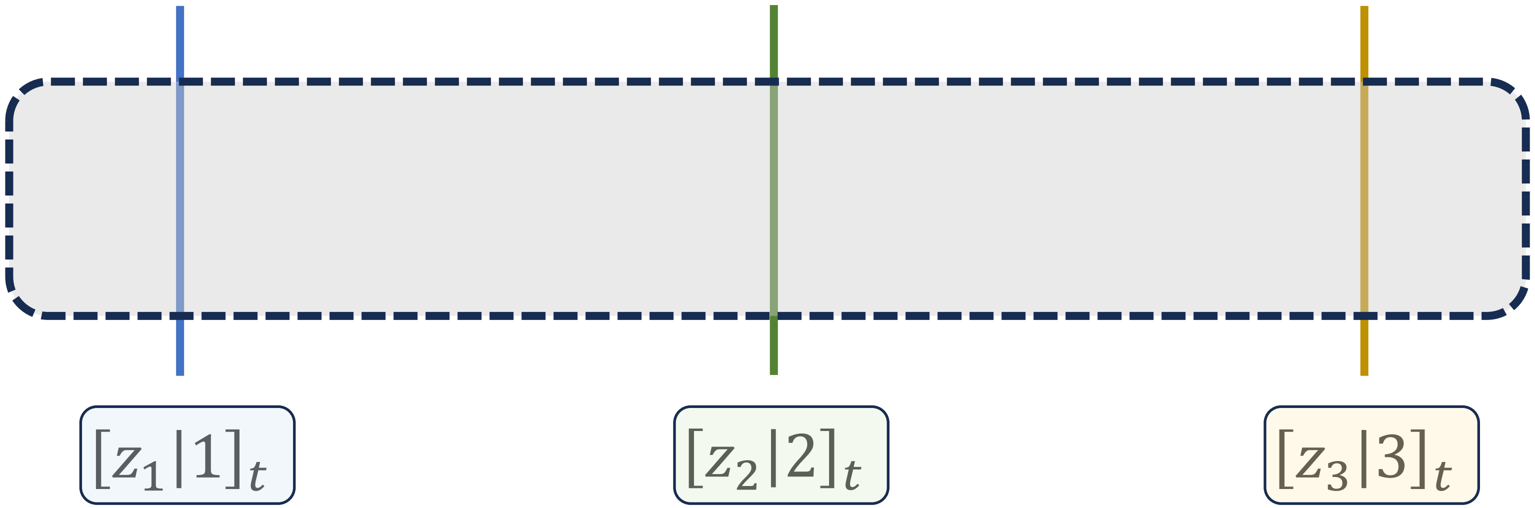
$[z_1\ z_2\ z_3]_{t+k-1}$



Towards GOD: Identifying Dispute Pairs - 1

Locally transform k degree-t sharings to packed sharings.

$[z_1\ z_2\ z_3]_{t+k-1}$



Towards GOD: Identifying Dispute Pairs - 1

Locally transform k degree-t sharings to packed sharings.

$$\boxed{[z_1\ z_2\ z_3]_{t+k-1}} = [\boldsymbol{e}_1]_{k-1} \cdot \boxed{[z_1|1]_t} + [\boldsymbol{e}_2]_{k-1} \cdot \boxed{[z_2|2]_t} + [\boldsymbol{e}_3]_{k-1} \cdot \boxed{[z_3|3]_t}$$

Towards GOD: Identifying Dispute Pairs - 1

Locally transform k degree-t sharings to packed sharings.

$$[z_1\ z_2\ z_3]_{t+k-1} = [\boldsymbol{e}_1]_{k-1} \cdot [z_1|1]_t + [\boldsymbol{e}_2]_{k-1} \cdot [z_2|2]_t + [\boldsymbol{e}_3]_{k-1} \cdot [z_3|3]_t$$

Towards GOD: Identifying Dispute Pairs - 1

Locally transform k degree-t sharings to packed sharings.

$$[z_1\ z_2\ z_3]_{t+k-1} = [\boldsymbol{e}_1]_{k-1} \cdot [z_1|1]_t + [\boldsymbol{e}_2]_{k-1} \cdot [z_2|2]_t + [\boldsymbol{e}_3]_{k-1} \cdot [z_3|3]_t$$

Error correction

A dispute pair

Recall

$4k - 2 = 2t + 2$

Towards GOD: Double-dipping Issue [GLS19]

t corrupted

$2k - 1$ honest

$2k - 1$ honest

$[\boldsymbol{x} + \boldsymbol{a}]_{t+k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{t+k-1}$

1

$[\boldsymbol{e}']_{k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{k-1}$

$[\boldsymbol{x} + \boldsymbol{a}]_{k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{k-1}$

2

$[\boldsymbol{z}' + \boldsymbol{r}]_{t+2k-2}$

$[\boldsymbol{z} + \boldsymbol{r}]_{t+2k-2}$

3

$(\boldsymbol{z}' + \boldsymbol{r}) - (\boldsymbol{z} + \boldsymbol{r}) = \left(\boldsymbol{e}' - (\boldsymbol{x} + \boldsymbol{a})\right) * \boldsymbol{y}$

Simpler method

Outline

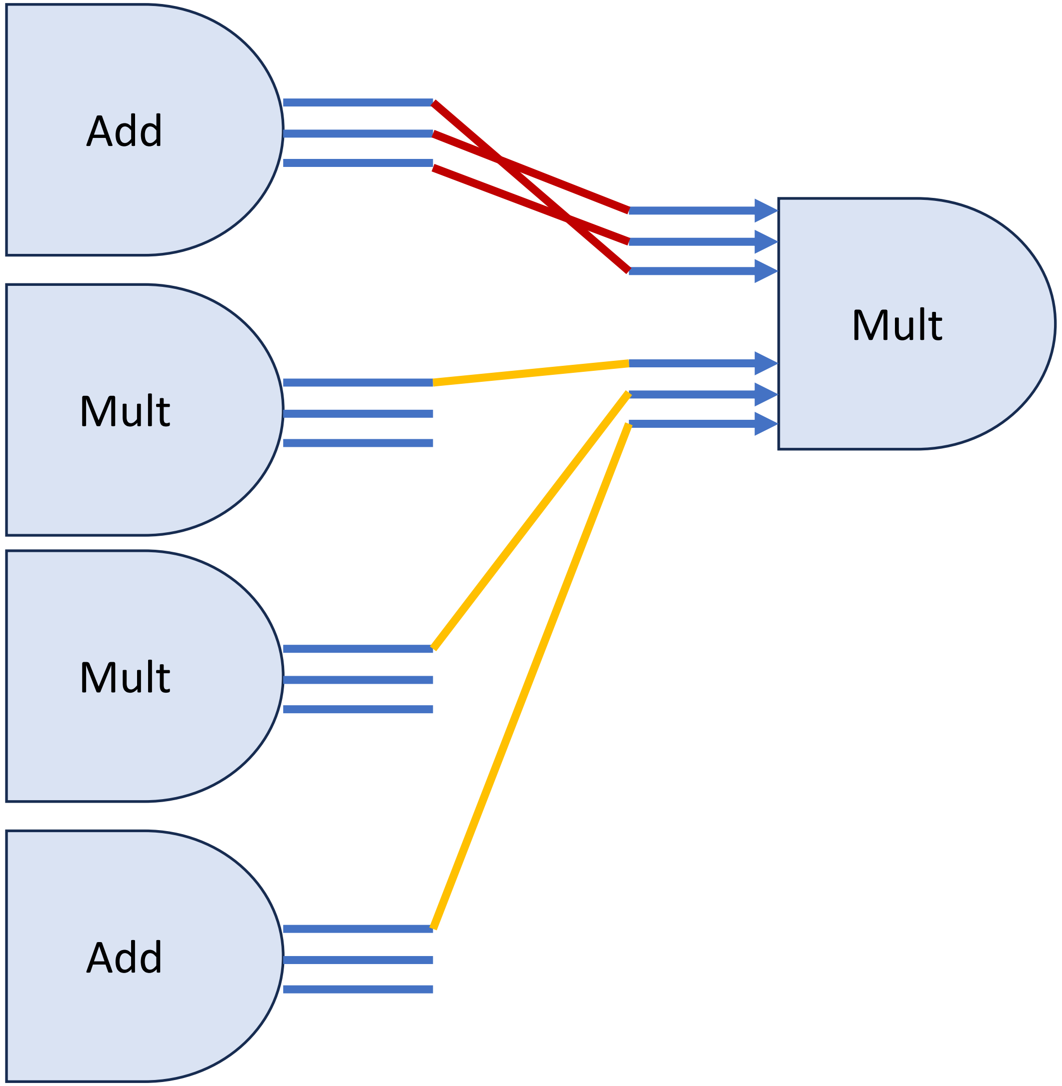
- Review: semi-honest protocol in [EGPS22]

- Towards full security via dispute control:

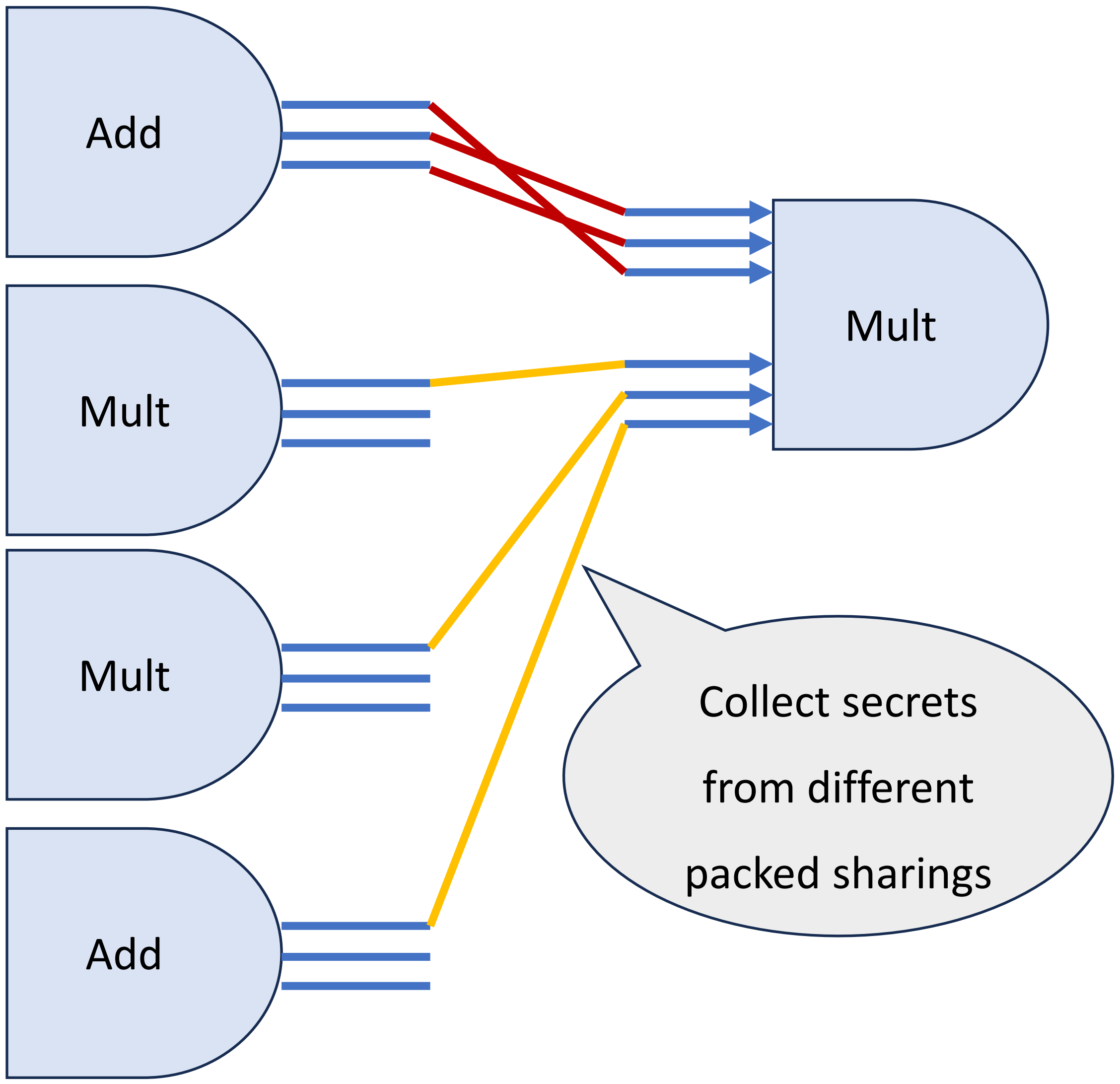
  verification + identifying dispute pairs

- <u>**Towards general circuits via sharing transformation**</u>

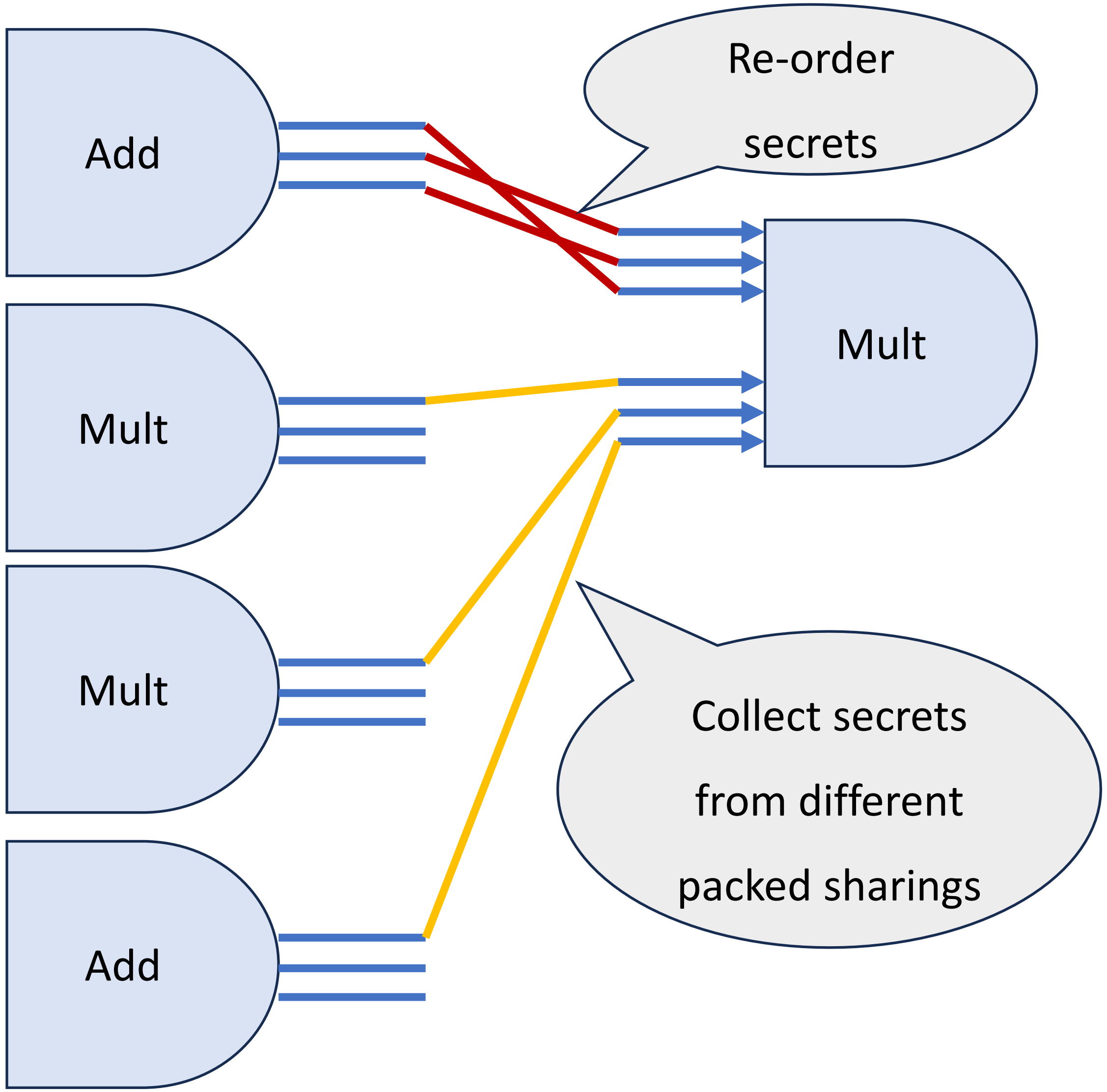
Towards general circuits: Network Routing [GPS21]



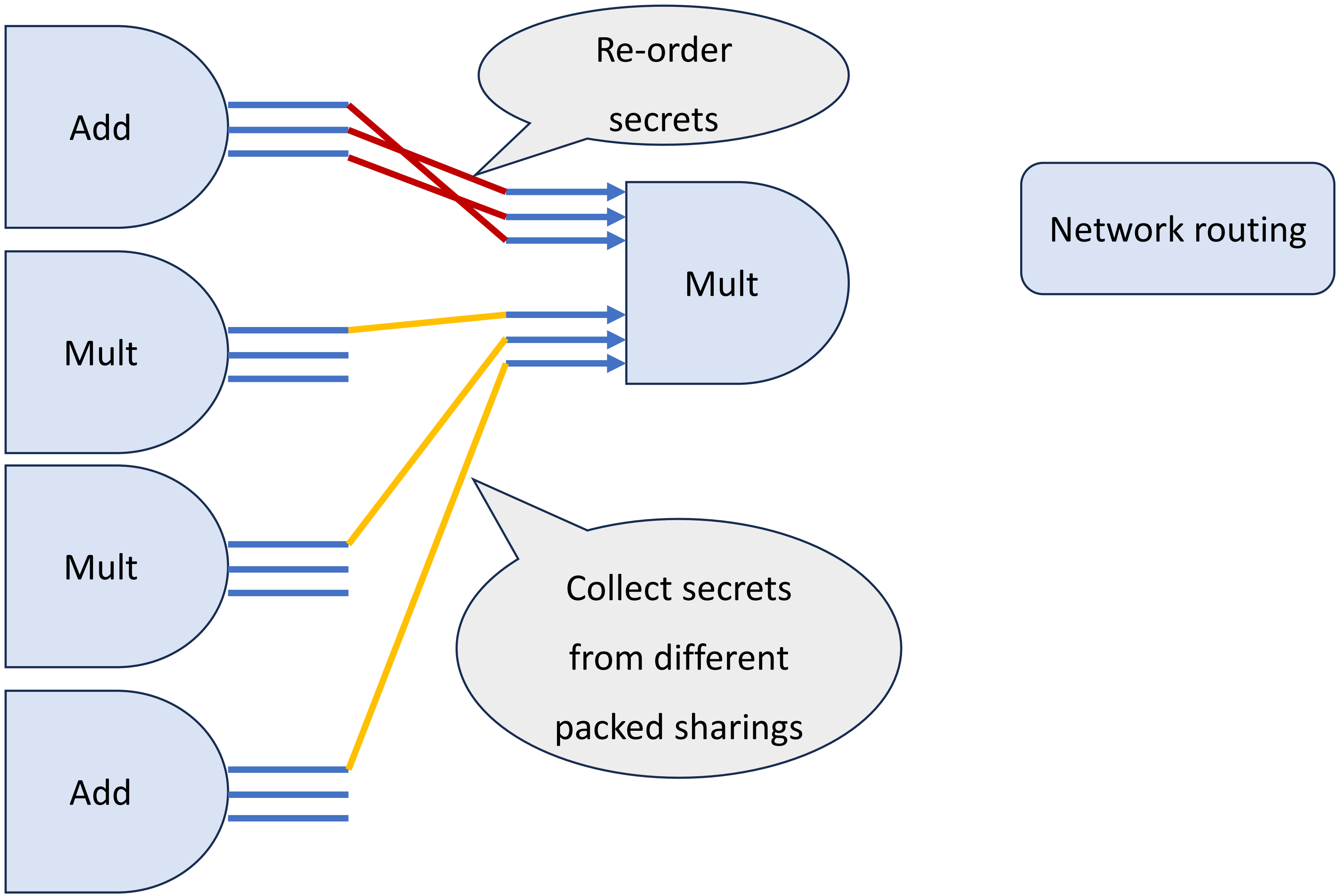
Towards general circuits: Network Routing [GPS21]



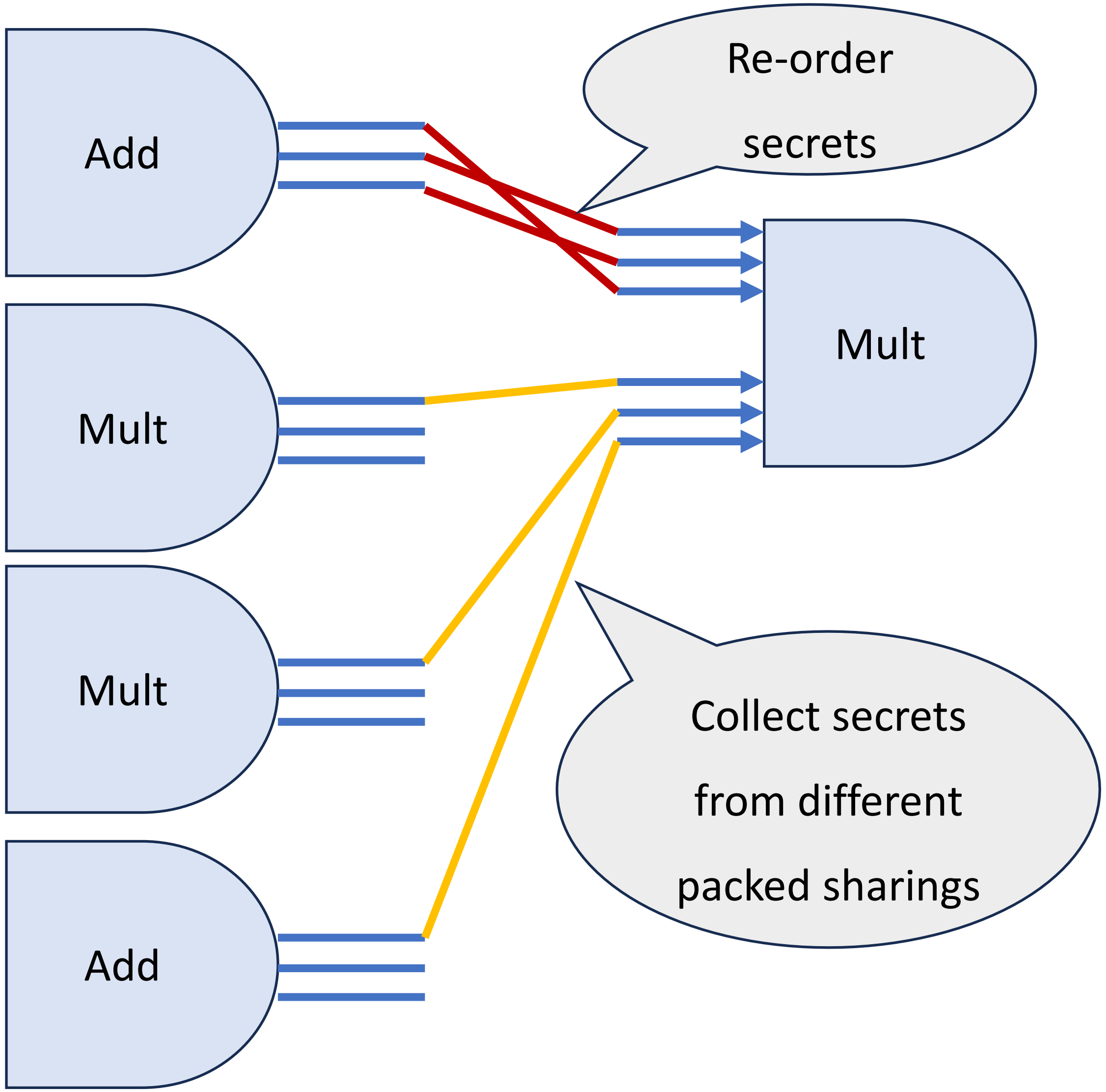
Towards general circuits: Network Routing [GPS21]



Towards general circuits: Network Routing [GPS21]



Towards general circuits: Network Routing [GPS21]



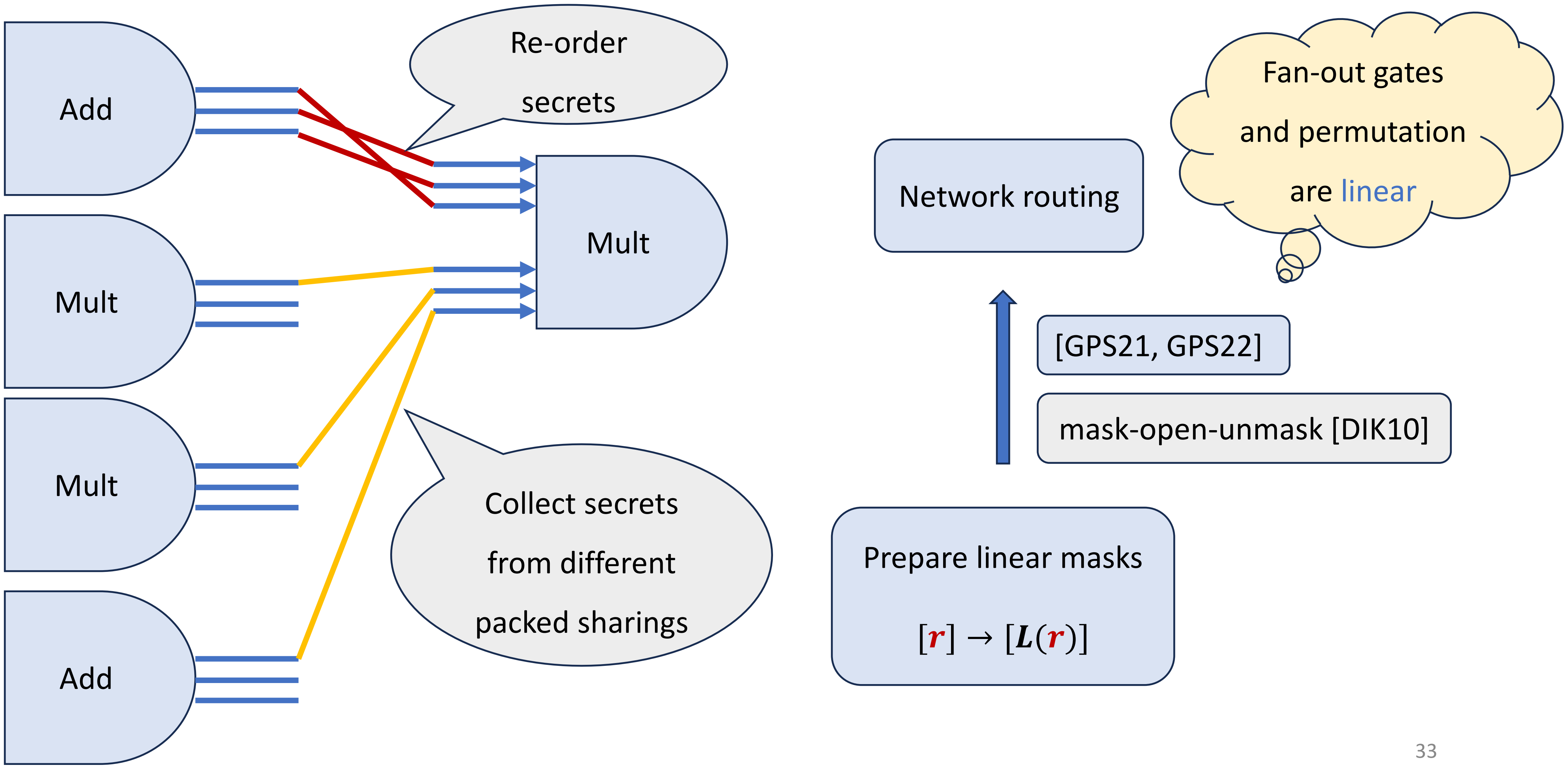
Network routing

Fan-out gates and permutation are linear

[GPS21, GPS22]

mask-open-unmask [DIK10]

Towards general circuits: Network Routing [GPS21]



Towards general circuits: Random linear mask

Goal: Prepare $[\boldsymbol{r}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$

$+$

Different linear transformations $\boldsymbol{L}$

Towards general circuits: Random linear mask

Goal: Prepare $[\boldsymbol{r}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$

$+$

Different linear transformations $\boldsymbol{L}$

$$[\boldsymbol{r}]_{t+k-1} \quad \boxed{r_1} \boxed{r_2} \boxed{r_3} \xrightarrow{\boldsymbol{L} = (L_1, L_2, L_3)} \boxed{L_1(\boldsymbol{r})} \boxed{L_2(\boldsymbol{r})} \boxed{L_3(\boldsymbol{r})} \quad [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$$

Towards general circuits: Random linear mask

Goal: Prepare $[\boldsymbol{r}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$

$+$

Different linear transformations $\boldsymbol{L}$

$[\boldsymbol{r}]_{t+k-1}$	r_1	r_2	r_3
	1st position	2nd position	3rd position

$$\boldsymbol{L} = (L_1, L_2, L_3) \Rightarrow$$

$L_1(\boldsymbol{r})$	$L_2(\boldsymbol{r})$	$L_3(\boldsymbol{r})$	$[\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$
1st position	2nd position	3rd position	

Towards general circuits: Random linear mask

$\boldsymbol{L} = (L_1, L_2, L_3)$

$[r_1 \mid 1]_t$	r_1	$L_1(\boldsymbol{r})$	$[L_1(\boldsymbol{r}) \mid 1]_t$
$[r_2 \mid 1]_t$	r_2	$L_2(\boldsymbol{r})$	$[L_2(\boldsymbol{r}) \mid 1]_t$
$[r_3 \mid 1]_t$	r_3	$L_3(\boldsymbol{r})$	$[L_3(\boldsymbol{r}) \mid 1]_t$
	1^{st} position	1^{st} position	

Towards general circuits: Random linear mask

$\boldsymbol{L} = (L_1, L_2, L_3)$

$[r_1\|1]_t$	r_1		$L_1(\boldsymbol{r})$	$[L_1(\boldsymbol{r})\|1]_t$
$[r_2\|1]_t$	r_2		$L_2(\boldsymbol{r})$	$[L_2(\boldsymbol{r})\|1]_t$
$[r_3\|1]_t$	r_3		$L_3(\boldsymbol{r})$	$[L_3(\boldsymbol{r})\|1]_t$

$$L_1(\boldsymbol{r}) = c_1 \cdot r_1 + c_2 \cdot r_2 + c_3 \cdot r_3$$

$$[L_1(\boldsymbol{r})|1]_t = c_1 \cdot [r_1|1]_t + c_2 \cdot [r_2|2]_t + c_3 \cdot [r_3|3]_t$$

1st position

1st position

Towards general circuits: Random linear mask

$\boldsymbol{L} = (L_1, L_2, L_3)$

$[r_1\|1]_t$	r_1	$L_1(\boldsymbol{r})$	$[L_1(\boldsymbol{r})\|1]_t$
$[r_2\|1]_t$	r_2	$L_2(\boldsymbol{r})$	$[L_2(\boldsymbol{r})\|1]_t$
$[r_3\|1]_t$	r_3	$L_3(\boldsymbol{r})$	$[L_3(\boldsymbol{r})\|1]_t$
	1st position	1st position	

$$L_1(\boldsymbol{r}) = c_1 \cdot r_1 + c_2 \cdot r_2 + c_3 \cdot r_3$$

$$[L_1(\boldsymbol{r})|1]_t = c_1 \cdot [r_1|1]_t + c_2 \cdot [r_2|2]_t + c_3 \cdot [r_3|3]_t$$

Observation: linear transformation can be done **locally** if the secrets are stored at the **same position**.

Towards general circuits: Random linear mask

$[\boldsymbol{r_1}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r_1})]_{t+k-1}$

$[\boldsymbol{r_2}]_{t+k-1}, [\boldsymbol{\pi}(\boldsymbol{r_2})]_{t+k-1}$

$[\boldsymbol{r_3}]_{t+k-1}, [\boldsymbol{P}(\boldsymbol{r_3})]_{t+k-1}$

k transformations

Towards general circuits: Random linear mask

$[\boldsymbol{r_1}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r_1})]_{t+k-1}$

$[\boldsymbol{r_2}]_{t+k-1}, [\boldsymbol{\pi}(\boldsymbol{r_2})]_{t+k-1}$

$[\boldsymbol{r_3}]_{t+k-1}, [\boldsymbol{P}(\boldsymbol{r_3})]_{t+k-1}$

k transformations

	$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
k	$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
	$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Transpose $\Downarrow$

r_{11}	r_{21}	r_{31}
r_{12}	r_{22}	r_{32}
r_{13}	r_{23}	r_{33}

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Transpose $\Downarrow$

r_{11}	r_{21}	r_{31}
r_{12}		
r_{13}	r_{23}	r_{33}

Observation: linear transformation can be done **locally** if the secrets are stored at the **same position**.

Towards general circuits: Random linear mask

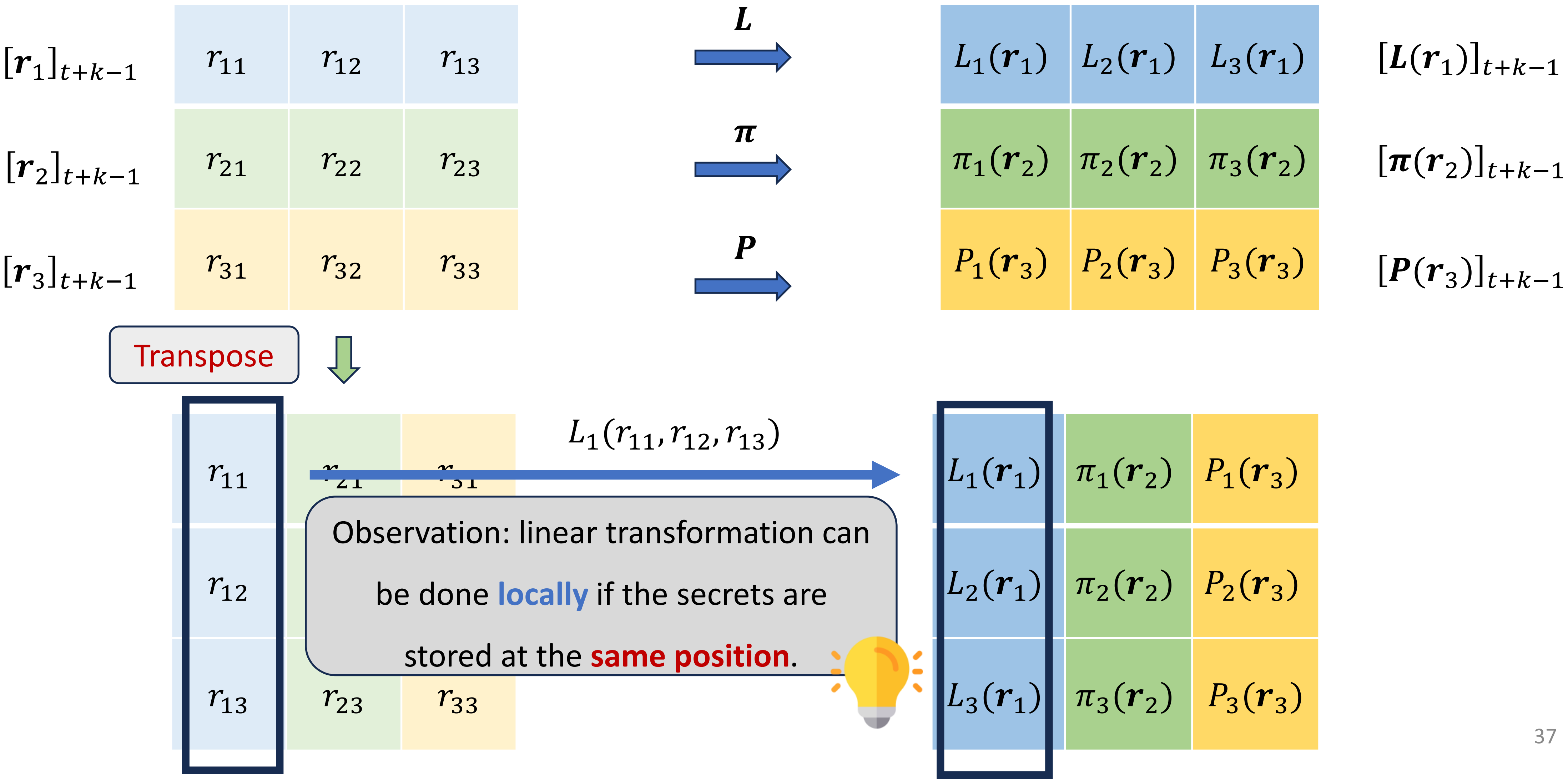
$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Transpose $\Downarrow$

r_{11}	r_{21}	r_{31}	$L_1(\boldsymbol{r}_1)$	$\pi_1(\boldsymbol{r}_2)$	$P_1(\boldsymbol{r}_3)$
r_{12}			$L_2(\boldsymbol{r}_1)$	$\pi_2(\boldsymbol{r}_2)$	$P_2(\boldsymbol{r}_3)$
r_{13}	r_{23}	r_{33}	$L_3(\boldsymbol{r}_1)$	$\pi_3(\boldsymbol{r}_2)$	$P_3(\boldsymbol{r}_3)$

Observation: linear transformation can be done **locally** if the secrets are stored at the **same position**.

Towards general circuits: Random linear mask



Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Transpose $\Downarrow$

r_{11}	r_{21}	r_{31}		$L_1(\boldsymbol{r}_1)$	$\pi_1(\boldsymbol{r}_2)$	$P_1(\boldsymbol{r}_3)$
r_{12}	r_{22}	r_{32}	Local comp $\Rightarrow$	$L_2(\boldsymbol{r}_1)$	$\pi_2(\boldsymbol{r}_2)$	$P_2(\boldsymbol{r}_3)$
r_{13}	r_{23}	r_{33}		$L_3(\boldsymbol{r}_1)$	$\pi_3(\boldsymbol{r}_2)$	$P_3(\boldsymbol{r}_3)$

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}	$\xrightarrow{\boldsymbol{L}}$	$L_1(\boldsymbol{r}_1)$	$L_2(\boldsymbol{r}_1)$	$L_3(\boldsymbol{r}_1)$	$[\boldsymbol{L}(\boldsymbol{r}_1)]_{t+k-1}$
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}	$\xrightarrow{\boldsymbol{\pi}}$	$\pi_1(\boldsymbol{r}_2)$	$\pi_2(\boldsymbol{r}_2)$	$\pi_3(\boldsymbol{r}_2)$	$[\boldsymbol{\pi}(\boldsymbol{r}_2)]_{t+k-1}$
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}	$\xrightarrow{\boldsymbol{P}}$	$P_1(\boldsymbol{r}_3)$	$P_2(\boldsymbol{r}_3)$	$P_3(\boldsymbol{r}_3)$	$[\boldsymbol{P}(\boldsymbol{r}_3)]_{t+k-1}$

Transpose $\Downarrow$ $\qquad$ $\Uparrow$ Transpose

r_{11}	r_{21}	r_{31}		$L_1(\boldsymbol{r}_1)$	$\pi_1(\boldsymbol{r}_2)$	$P_1(\boldsymbol{r}_3)$
r_{12}	r_{22}	r_{32}	Local comp $\Rightarrow$	$L_2(\boldsymbol{r}_1)$	$\pi_2(\boldsymbol{r}_2)$	$P_2(\boldsymbol{r}_3)$
r_{13}	r_{23}	r_{33}		$L_3(\boldsymbol{r}_1)$	$\pi_3(\boldsymbol{r}_2)$	$P_3(\boldsymbol{r}_3)$

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}

Transpose

r_{11}	r_{21}	r_{31}
r_{12}	r_{22}	r_{32}
r_{13}	r_{23}	r_{33}

Prepare $[\boldsymbol{r}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$

Perform '**transpose**' operation

Towards general circuits: Random linear mask

$[\boldsymbol{r}_1]_{t+k-1}$	r_{11}	r_{12}	r_{13}
$[\boldsymbol{r}_2]_{t+k-1}$	r_{21}	r_{22}	r_{23}
$[\boldsymbol{r}_3]_{t+k-1}$	r_{31}	r_{32}	r_{33}

Transpose

r_{11}	r_{21}	r_{31}
r_{12}	r_{22}	r_{32}
r_{13}	r_{23}	r_{33}

Prepare $[\boldsymbol{r}]_{t+k-1}, [\boldsymbol{L}(\boldsymbol{r})]_{t+k-1}$

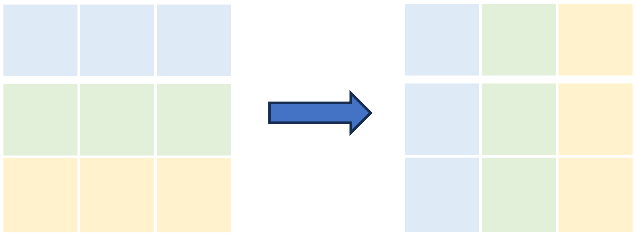
Perform ‘**transpose**’ operation

Transpose is a special linear transformation

Towards general circuits: Random linear mask

Perform 'transpose' operation

$$\left\{ [r_{ij} | j]_t, [r_{ij} | i]_t \right\}_{i,j}$$



Towards general circuits: Random linear mask

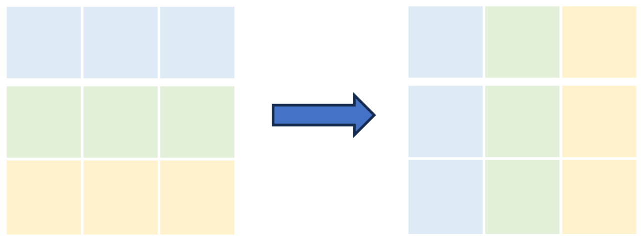
Perform ‘transpose’ operation

$$\left\{[r_{ij}|j]_t, [r_{ij}|i]_t\right\}_{i,j}$$

mask-open-unmask

Prepare random linear masks for ‘transpose’

$\left\{[u_{ij}|j]_t, [u_{ij}|i]_t\right\}_{i,j}$ **in the preprocessing phase.**



Towards general circuits: Random linear mask

Perform ‘transpose’ operation

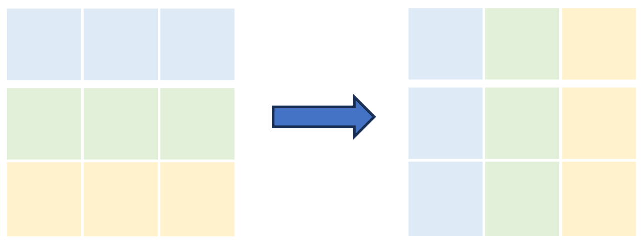
$$\left\{[r_{ij}|j]_t, [r_{ij}|i]_t\right\}_{i,j}$$

mask-open-unmask

Prepare random linear masks for ‘transpose’

$\left\{[u_{ij}|j]_t, [u_{ij}|i]_t\right\}_{i,j}$ **in the preprocessing phase.**

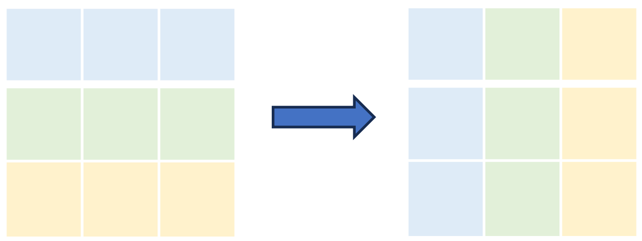
Known extraction technique



Towards general circuits: Random linear mask

Perform ‘transpose’ operation

$\left\{ [{\color{red}r_{ij}}|j]_{t'}, [{\color{red}r_{ij}}|i]_{t} \right\}_{i,j}$



mask-open-unmask

Prepare random linear masks for ‘transpose’

$\left\{ [{\color{red}u_{ij}}|j]_{t'}, [{\color{red}u_{ij}}|i]_{t} \right\}_{i,j}$ **in the preprocessing phase.**

Online communication remains $O(n)$ per linear transformation.

Known extraction technique

Summary

Semi-honest protocol from [EGPS22]

Summary

Semi-honest protocol from [EGPS22]

Cross-layer multiplication

Add deg-$(n' - 1)$ sharings and open to prevent **double-dipping**

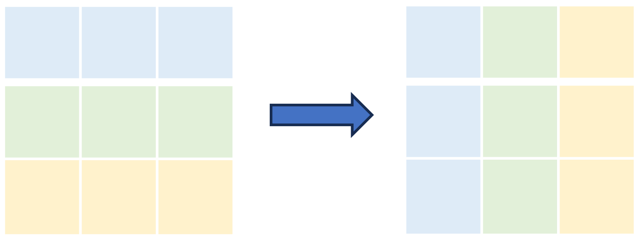
Summary

Semi-honest protocol from [EGPS22]

Cross-layer multiplication

Efficient sharing transformation

Reduce **different** linear transformations to ‘**transpose**’



Summary

Semi-honest protocol from [EGPS22]

Cross-layer multiplication

Efficient sharing transformation

Dispute control

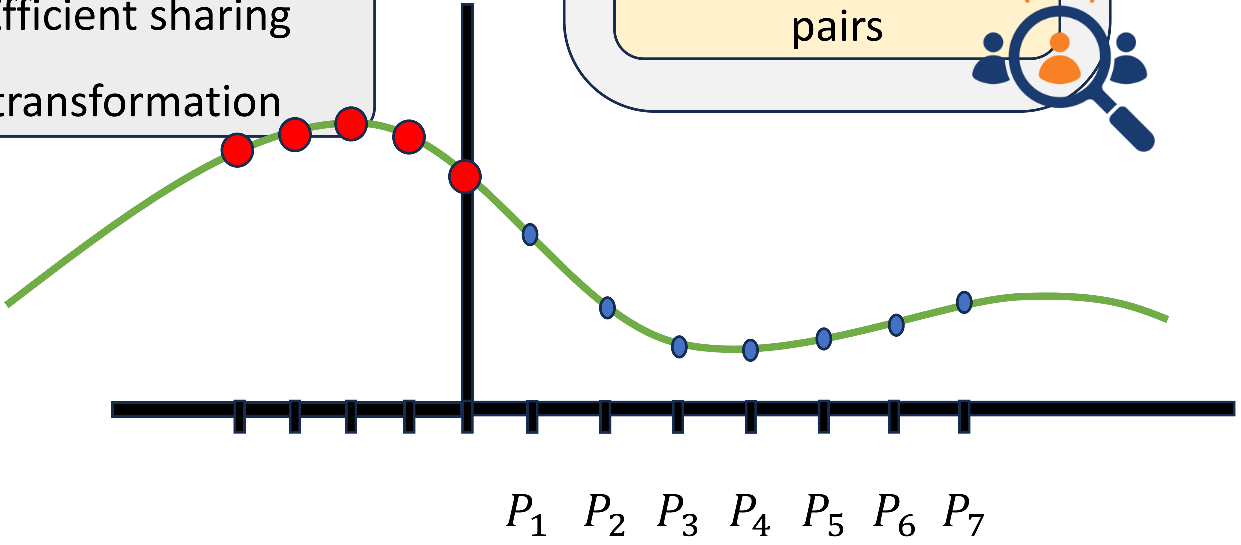
- Efficient verification
- Identify dispute pairs

Summary

Semi-honest protocol from [EGPS22]

Cross-layer multiplication

Efficient sharing transformation



Dispute control

Efficient verification

Identify dispute pairs

Summary

Semi-honest protocol from [EGPS22]

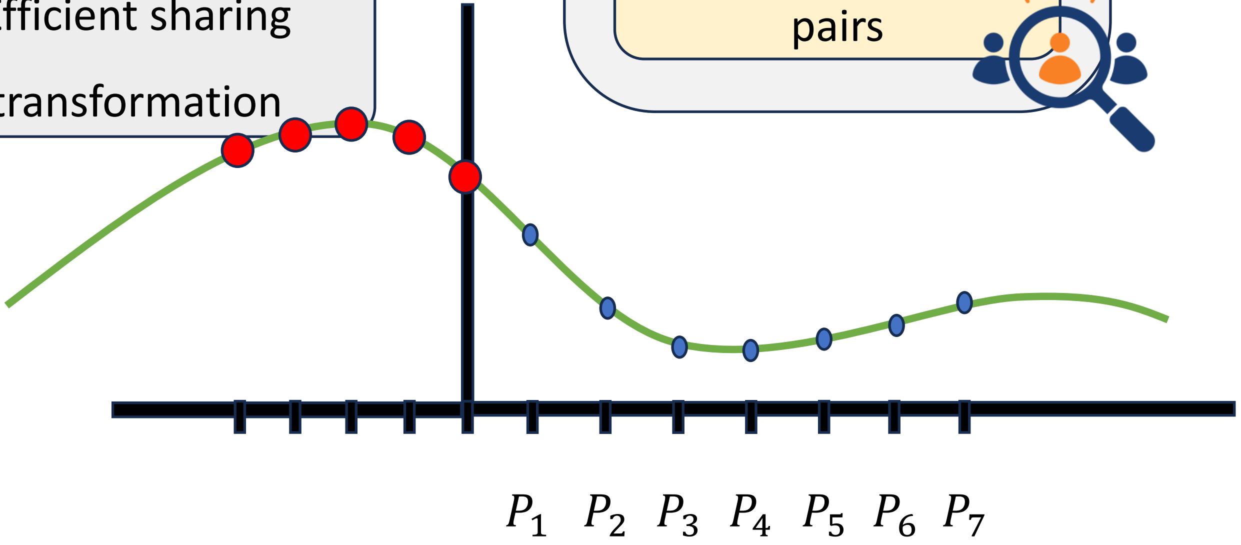
Cross-layer multiplication

Efficient sharing transformation

Dispute control

Efficient verification

Identify dispute pairs



Higher-degree Shamir sharings make things difficult.

Summary

Semi-honest protocol from [EGPS22]

Cross-layer multiplication

Efficient sharing transformation

+

Dispute control

Efficient verification

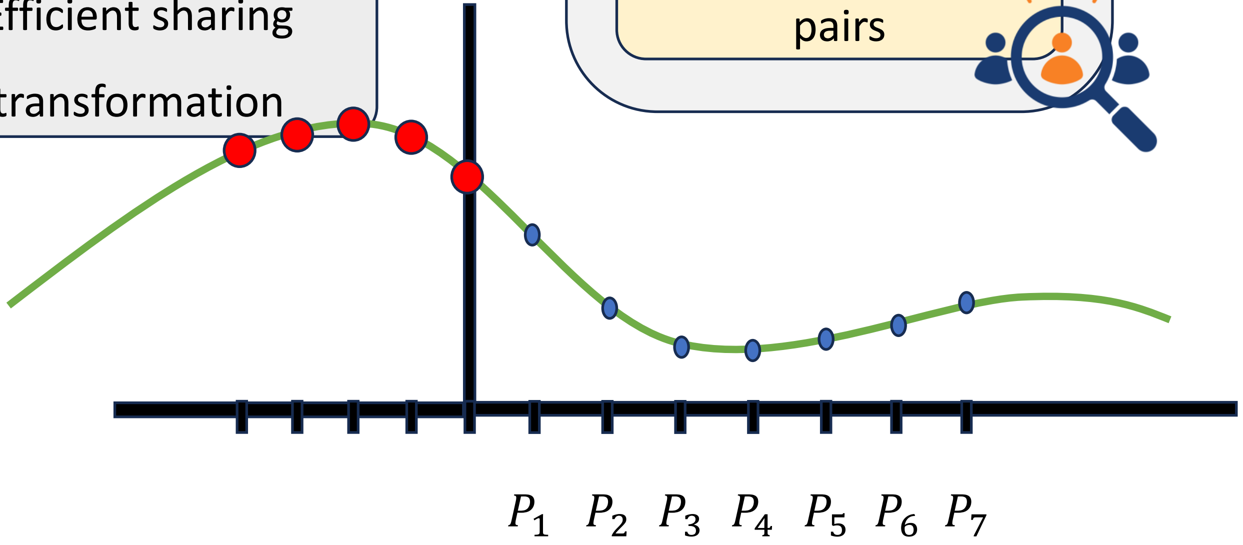
Identify dispute pairs

=

Perfectly-secure MPC

Online: $O(1)$ per gate

Offline: $O(n)$ per gate



Higher-degree Shamir sharings make things difficult.

Thank you!

Credit:
Icons: https://www.flaticon.com/