MPC in Large Networks

with Applications to Anonymous Broadcast

Mahdi Zamani
University of New Mexico

with Jared Saia and Mahnush Movahedi

Motivation

- Growth of modern networks

BitTorrent™ ~250 million users

facebook® ~1.2 billion users

twitter ~300 million users

bitcoin ~1.2 million users

Our Goal

- Practical MPC for large networks
  - From thousands to billions of parties
  - Malicious parties
  - Arithmetic functions
- Applications
  - Anonymous communication
  - Secure analysis of big data

MPC Approaches

- Compute a function f over
  - *Secret-shared* values
    - Shamir's sharing
    - Small computation cost
    - Multiplication requires several rounds.
  - *Encrypted* values
    - Fully Homomorphic Encryption (FHE)
    - Optimal round complexity
    - Has large computation cost.

Our Model

- n parties
  - Connected pairwise via private channels.
- $\leq n/10$ are malicious
  - Can deviate arbitrarily from our protocol.
- Adversary is
  - Computationally bounded
  - Static
- Synchronous communication

Our Results

- Average costs
  - Online phase
    - $O(m \log^3 n)$ messages of size $O(\log p)$
    - $O(m \log^4 n)$ operations
  - Offline phase
    - $\tilde{O}(n\,\kappa^2)$ messages and operations
  - $O(d)$ rounds of communication

Scalability *via* Quorums

- Logarithmic-size set of parties
- $< N/9$ malicious parties in each quorum
- Quorum building of [BGH'13]
- Used for MPC
  - Scalable MPC [DKMS'12]
  - Communication locality in MPC [BGT'13]

Building Blocks

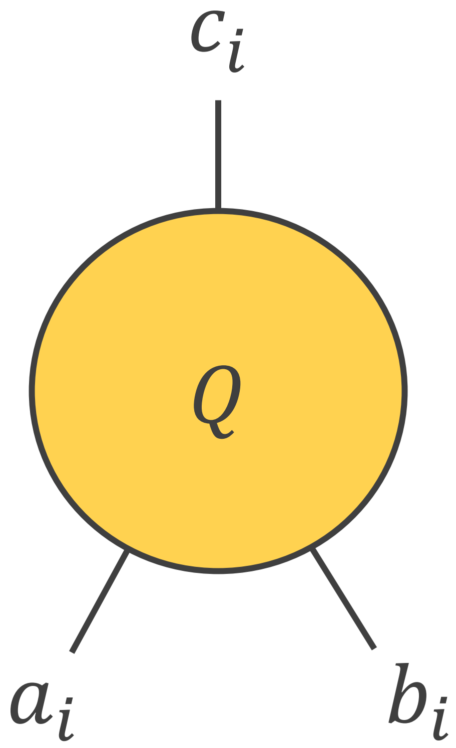
- Efficient VSS of [KZG'10]
  - Shamir's scheme along with commitments
- Threshold FHE of [AJTV'12]
  - Adopted [BGV'12] to the malicious case
- Preprocessing Model of [DPSZ'11]
  - FHE in offline phase

Our Protocol

- Create n quorums.
- Assign each gate G to a quorum Q.
- For each party $P_i \in Q$,
  - Compute G over secret-shared inputs.

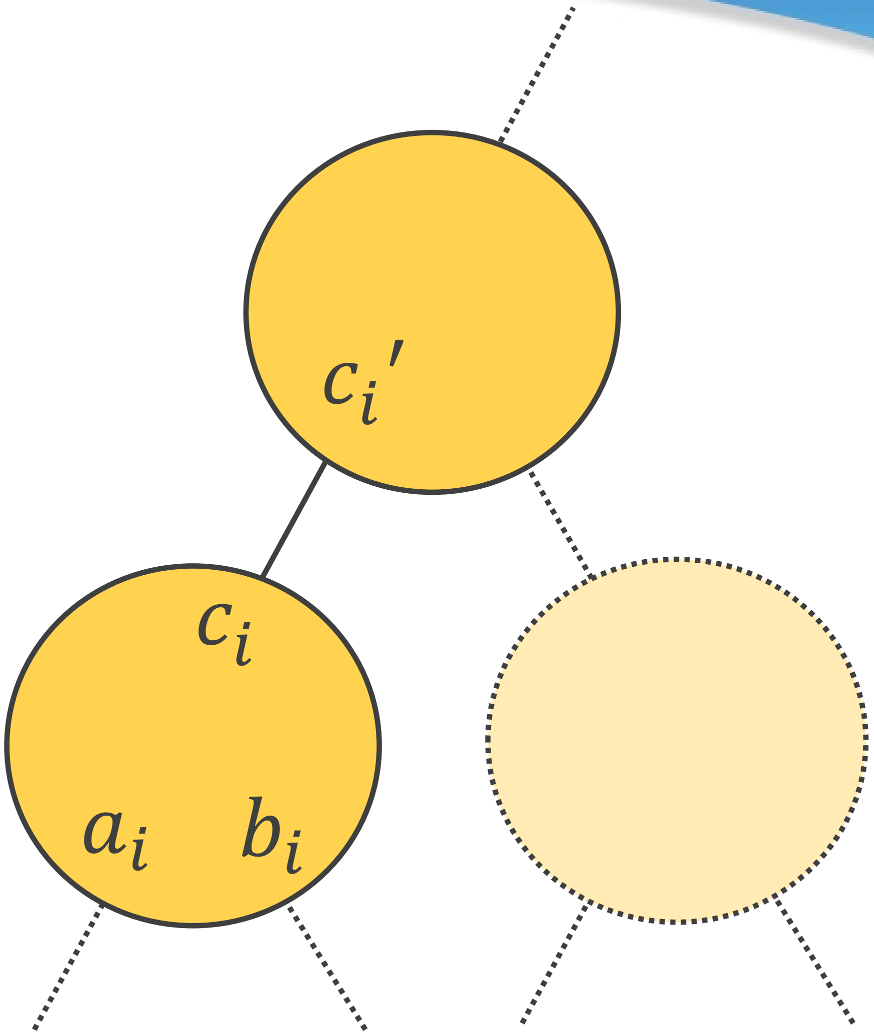
$$c_i = F_G(a_i, b_i)$$

$$c = F_G(a, b)$$



Challenges

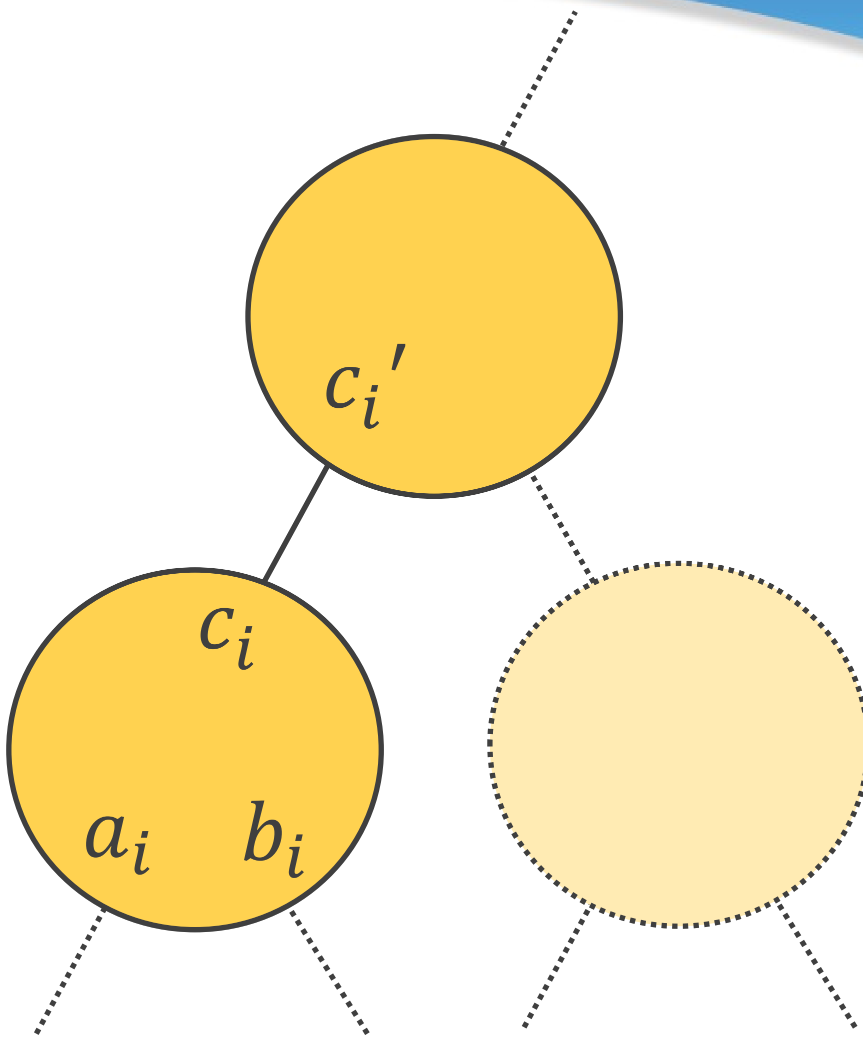
$$c_i = F_G(a_i, b_i)$$



Challenges

- Resharing

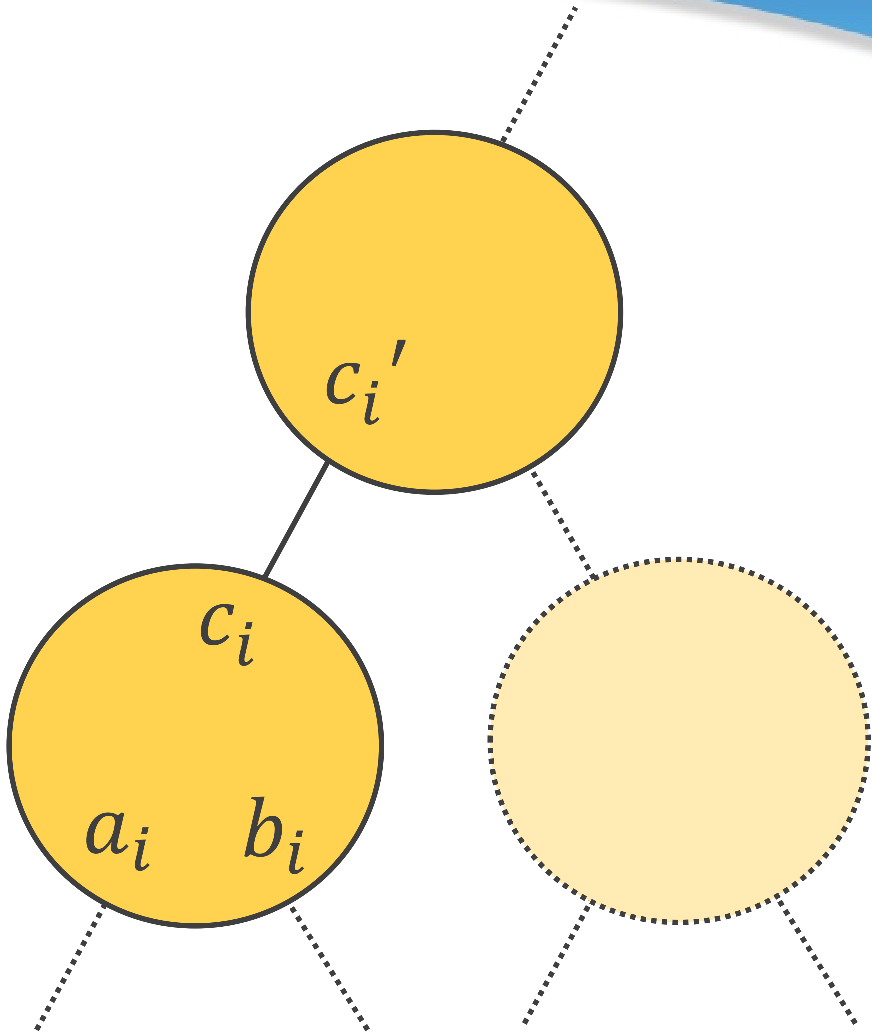
$$c_i = F_G(a_i, b_i)$$



Challenges

- Resharing
- Multiplication

$$c_i = F_G(a_i, b_i)$$



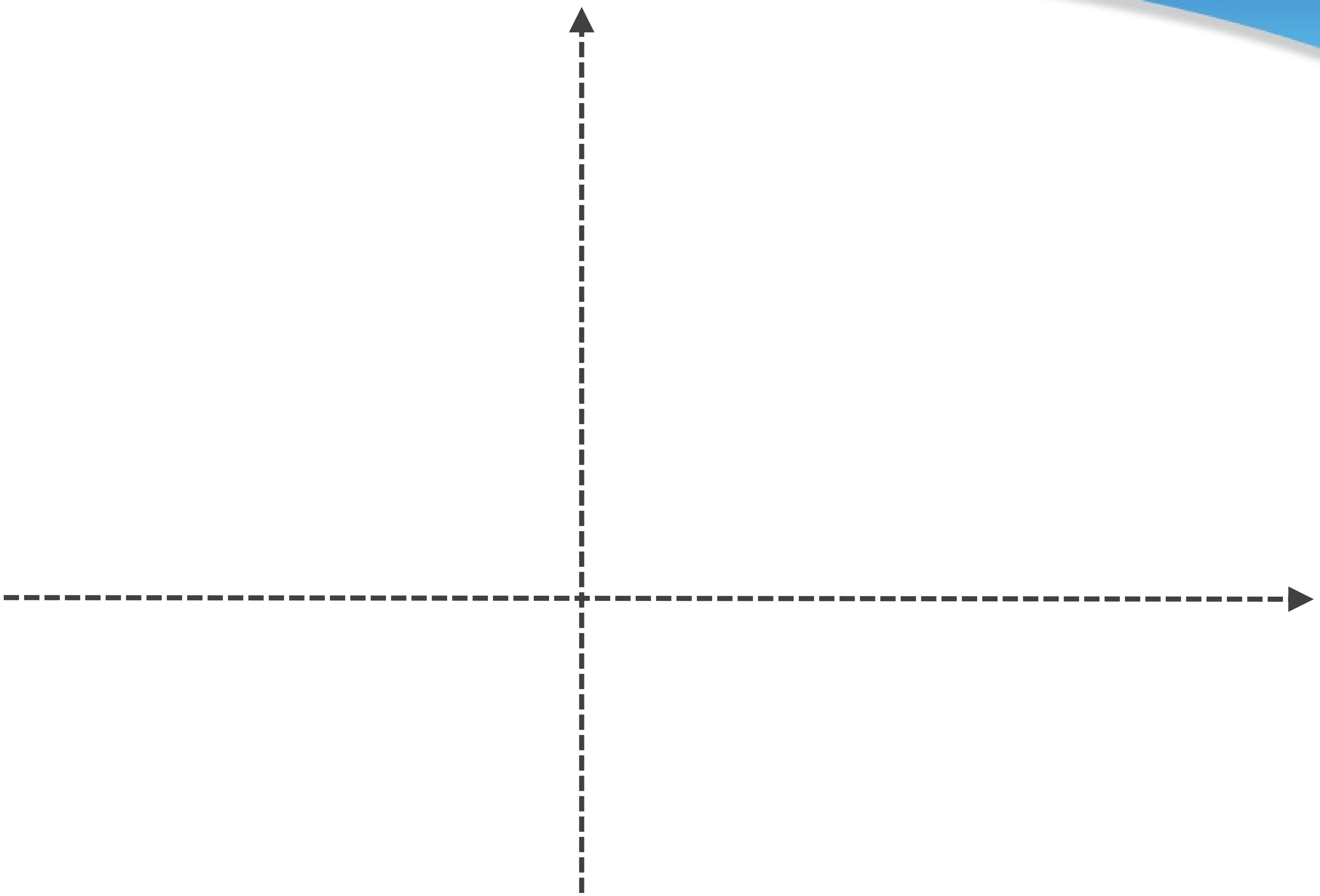
Resharing

- A Shamir sharing can be easily *refreshed.*
- A new polynomial with the same free term.

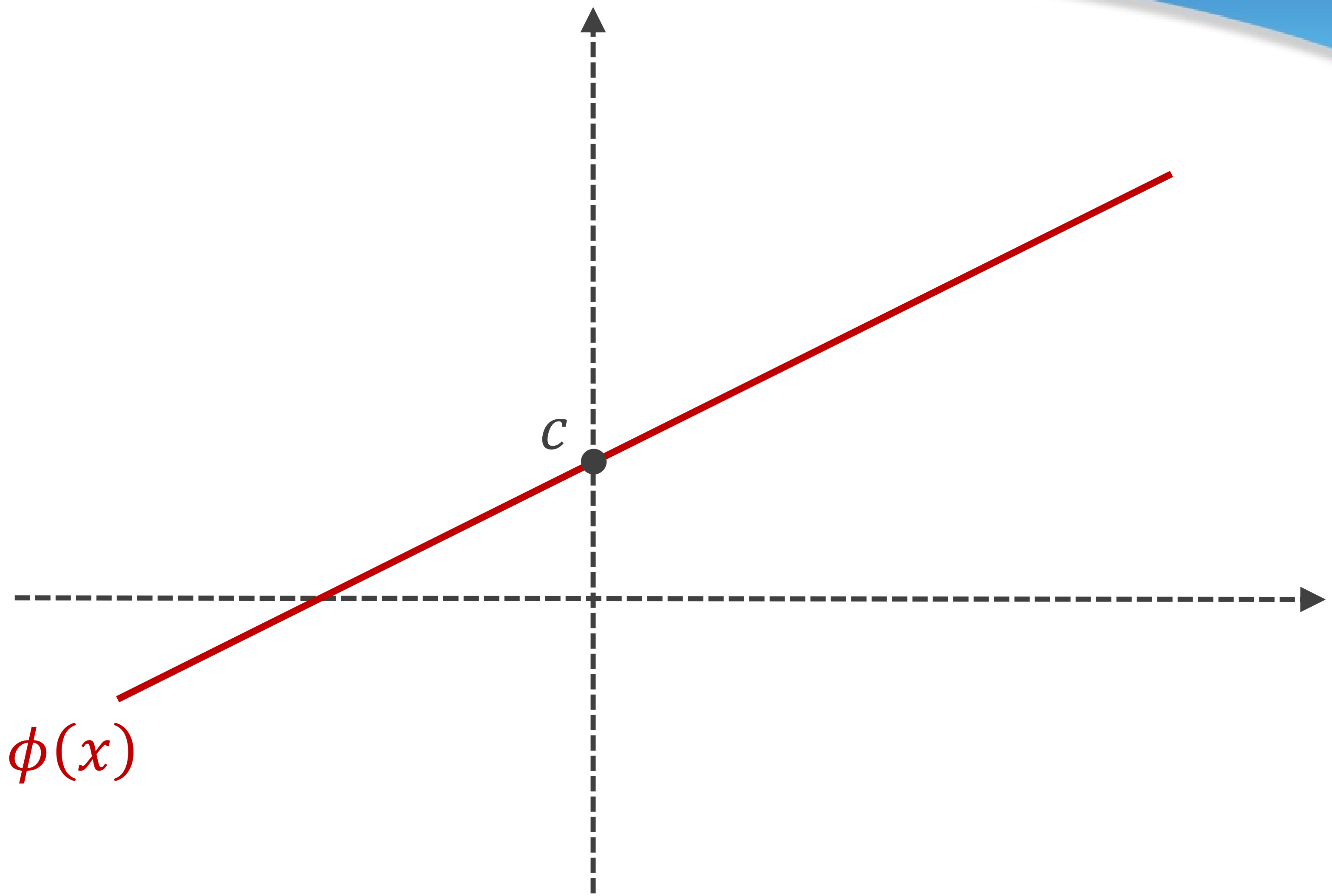
> “A frequent change of this type can greatly enhance security since the pieces exposed by security breaches cannot be accumulated unless all of them are values of the same edition of the polynomial.”

Adi Shamir. How to share a secret. 1979.

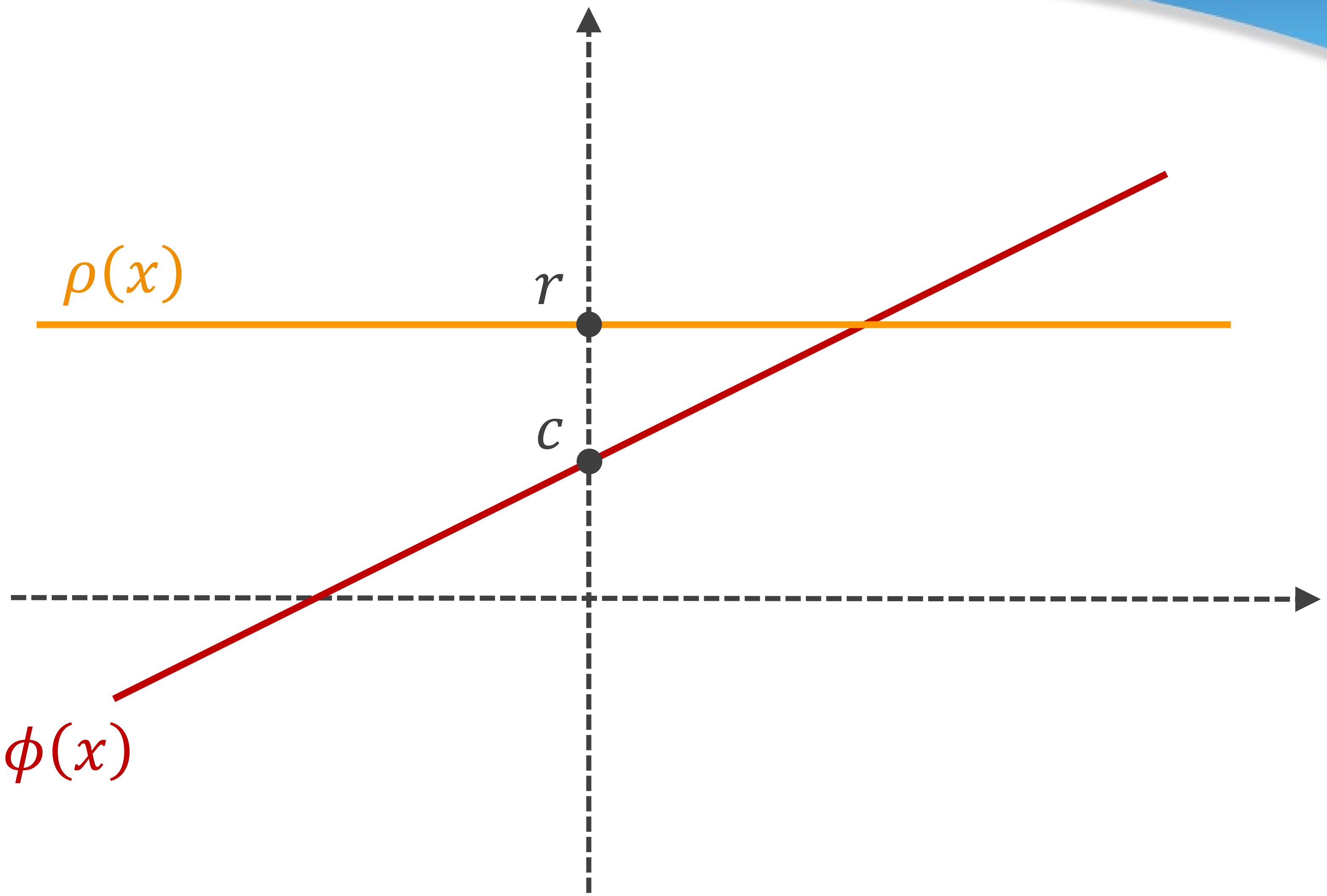
Resharing



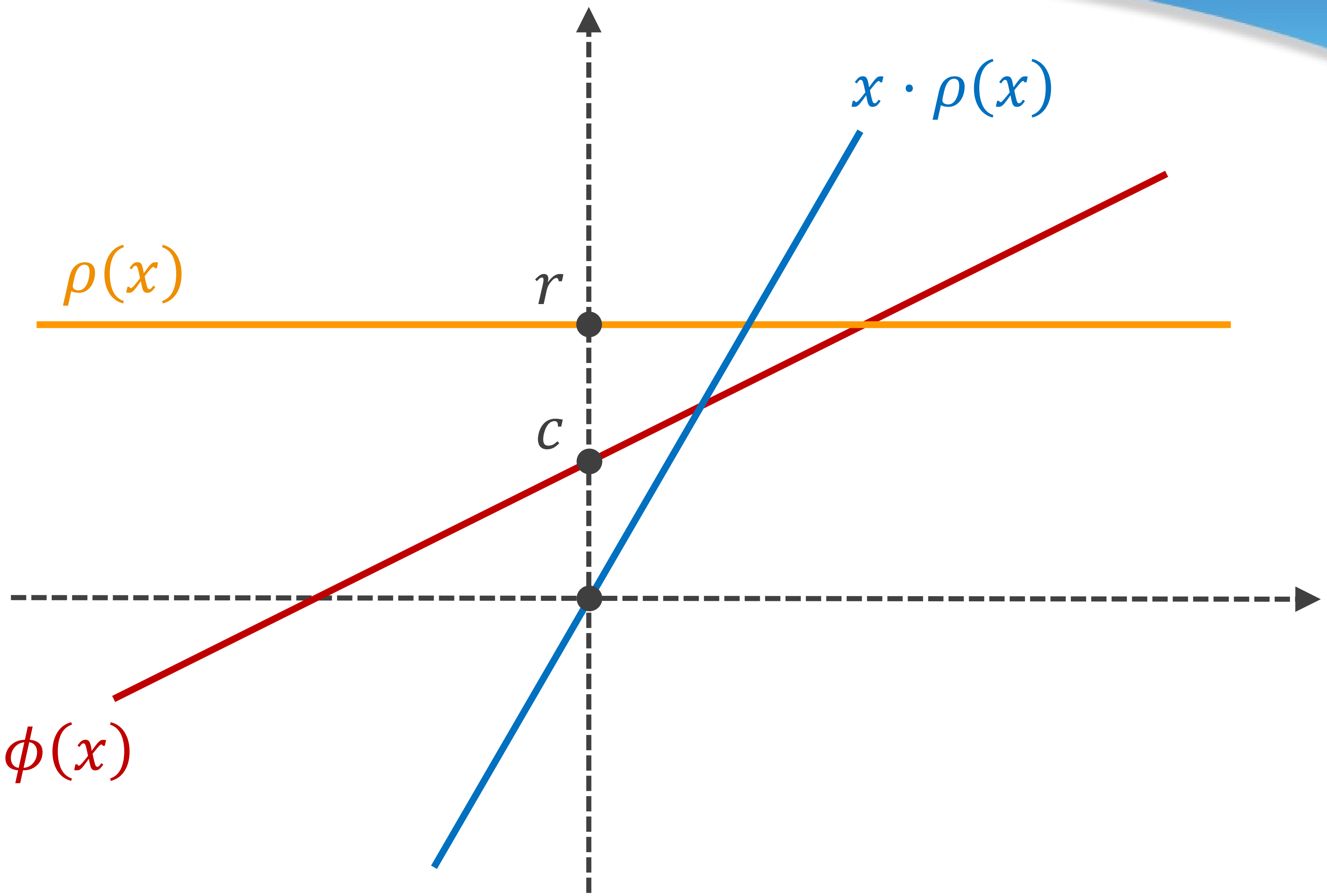
Resharing



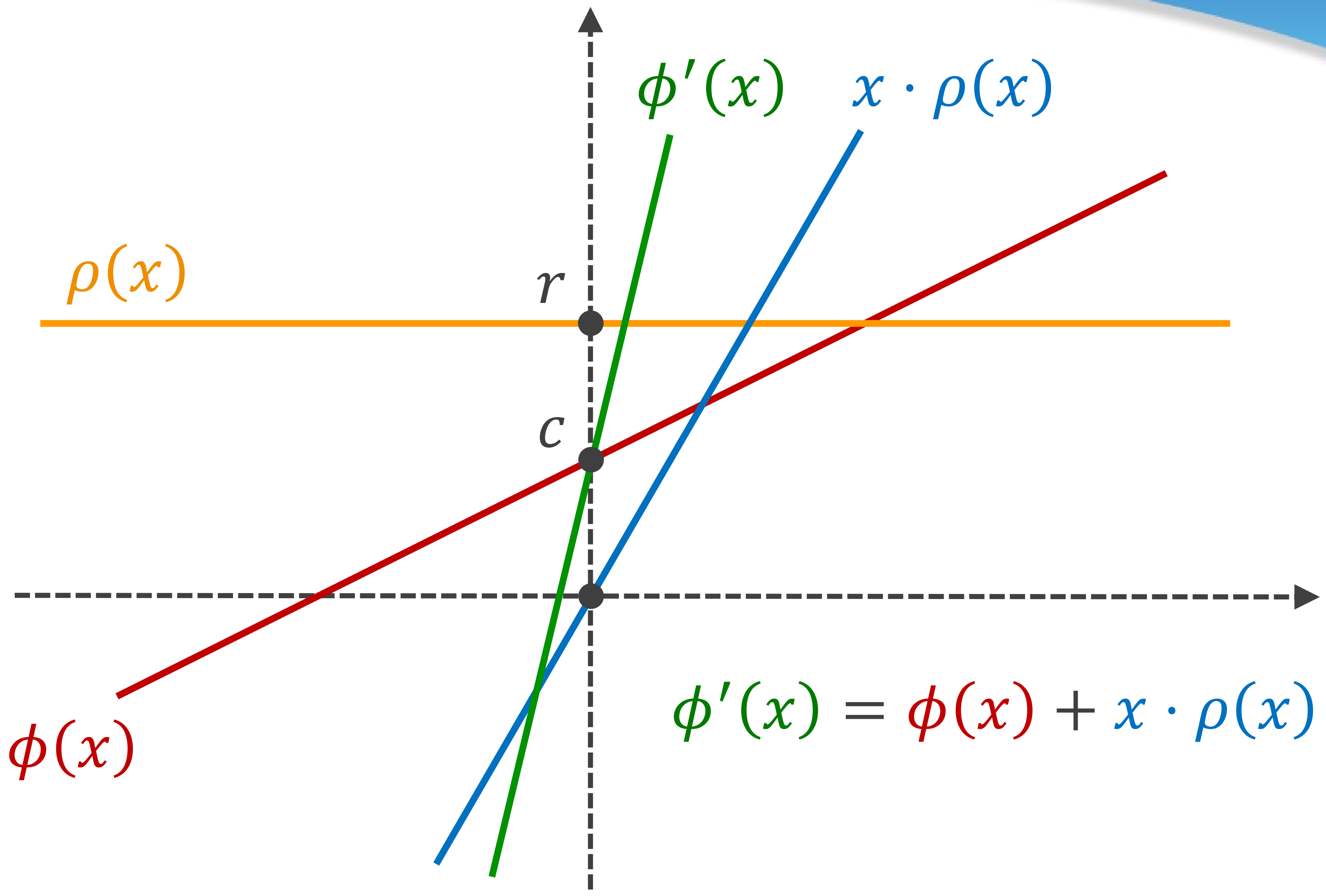
Resharing



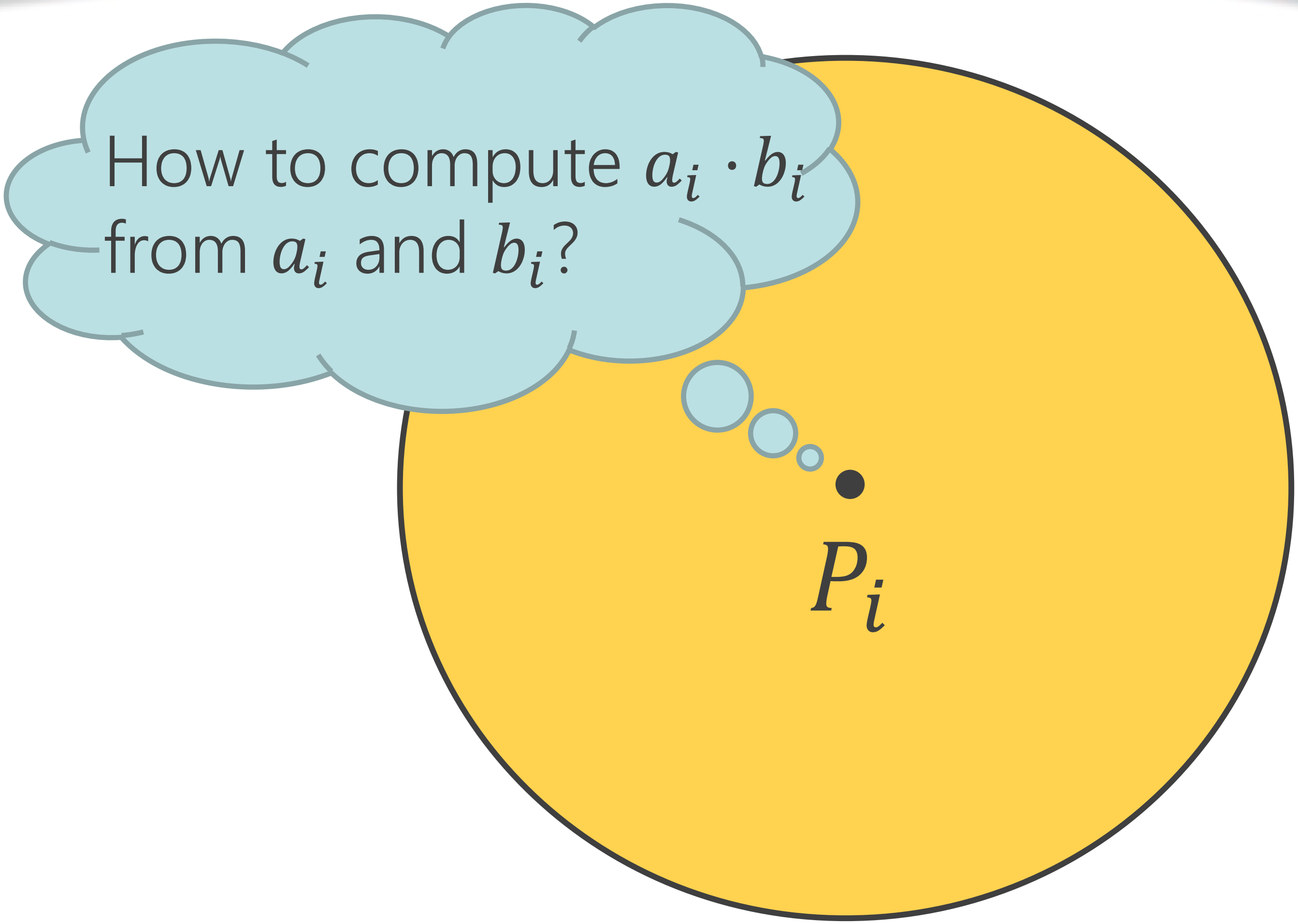
Resharing



Resharing



Multiplication over Shares



Beaver's Multiplication Triples

TFHE in
Offline phase

$$(u_i, v_i, w_i)$$

$$w = u \cdot v$$

Beaver's Multiplication Triples

In online phase

$$c_i = w_i - \delta a_i - \varepsilon b_i + \varepsilon\delta$$

$$\varepsilon_i = a_i + u_i$$

$$\delta_i = b_i + v_i$$

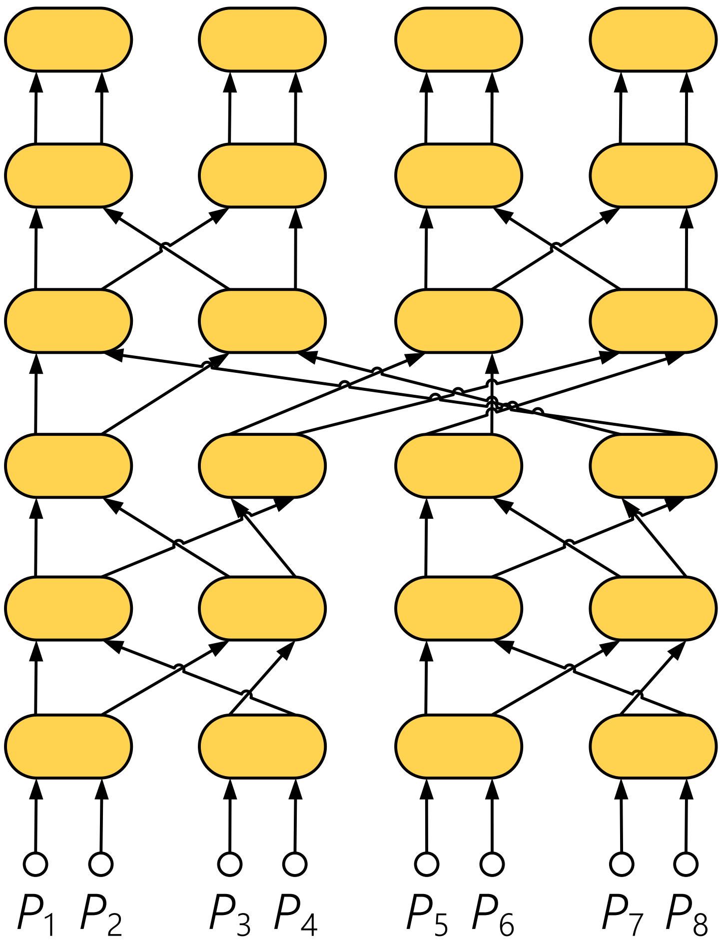
Application: Anonymous Broadcast

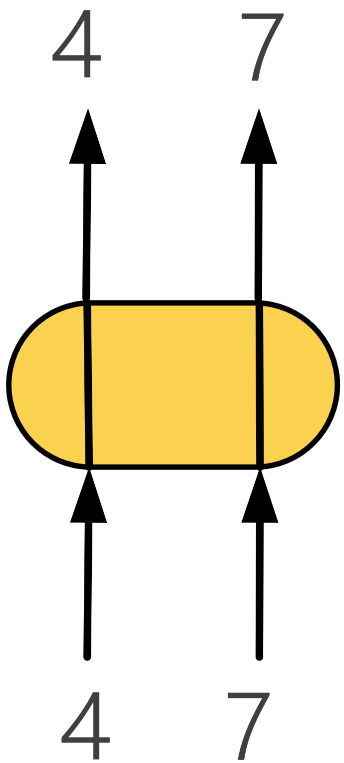
- Each party has a message to broadcast,
- No coalition should be able to map messages to senders.
- Current schemes are either vulnerable to traffic analysis or are impractical.

Anonymous Broadcast *via* MPS

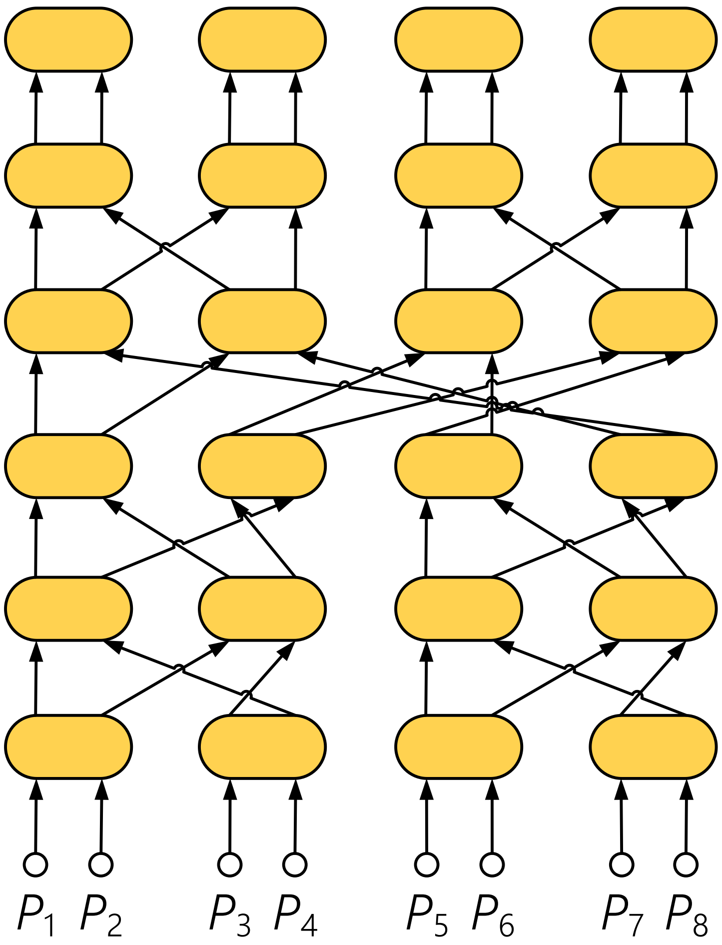
- Let m_i be P_i's message.
- P_i picks a random value r_i.
- Parties jointly *sort* their pairs (r_i, m_i) over r_i.
- Multi-Party Sorting (MPS)
  - Each party receives a vector of all sorted inputs.

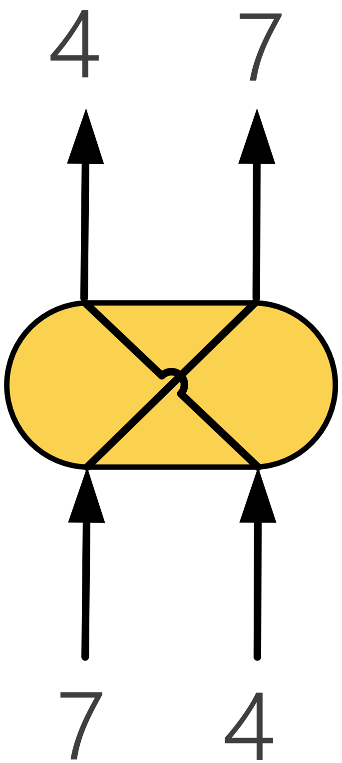
Multi-Party Sorting (MPS)



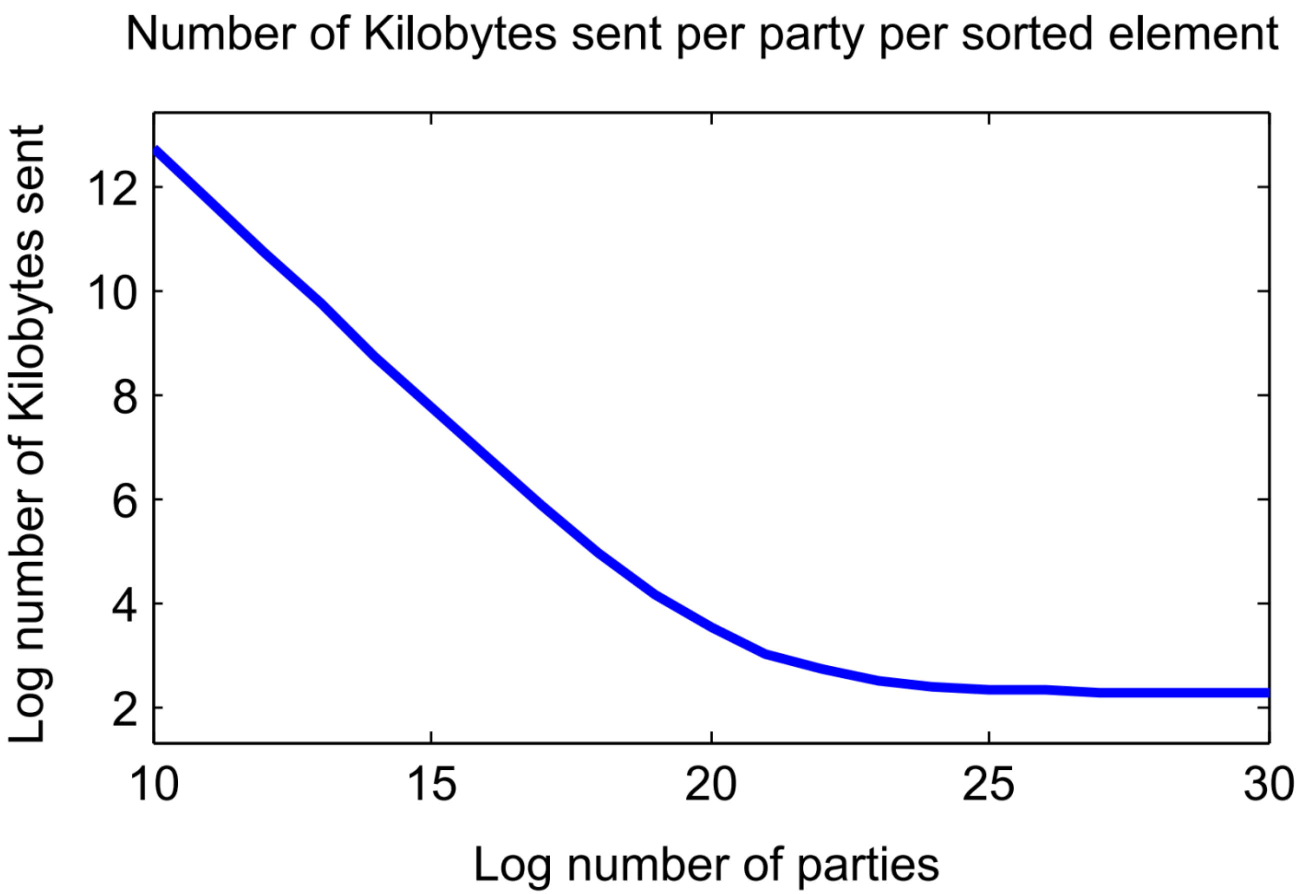


Multi-Party Sorting (MPS)

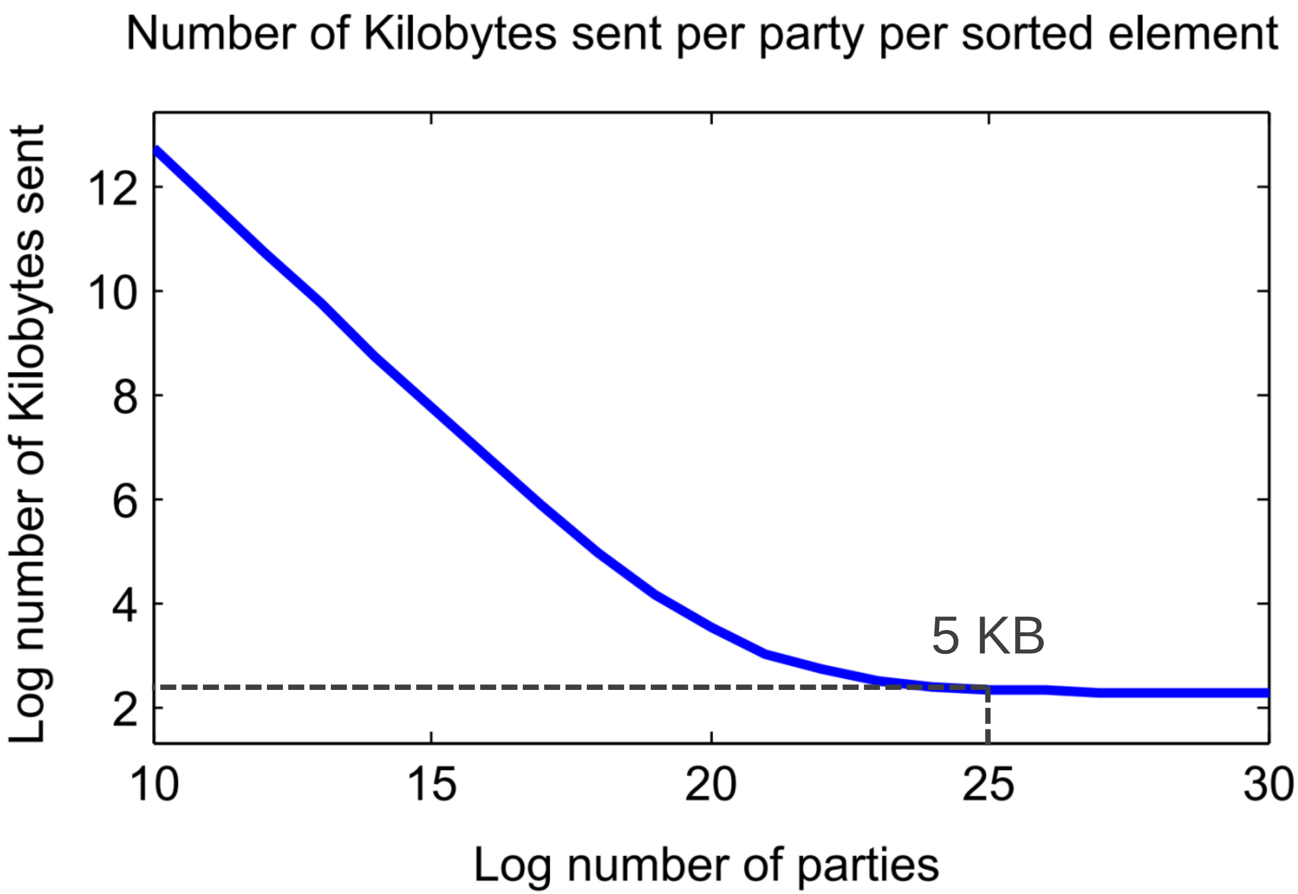




Microbenchmarks



Microbenchmarks



Conclusion

- An efficient protocol for MPC
- Tolerates up to $n/10$ malicious parties
- Efficient anonymous broadcast via multiparty sorting.

Open Problems

- Blacklist bad parties over time
- Asynchronous communication
- Adaptive adversary

Thank you!

Questions?

Costs Breakdown

- One sorting ($n = 2^{25}$)

Phase	**% phase**	**% total**
Setup		76%
Quorum building	5%	
Key generation	95%	
Triple generation	<1%	
Online		24%
Circuit computation	1%	
Output propagation	99%*	

Costs Breakdown

Phase	$n = 2^{10}$	$n = 2^{30}$
Setup	99%	8%
Online	1%	92%

Cost per Party

Each party sends about 160 GB for sorting 600 MB of data.

Overhead = 1:250

Protocol Recap

- Setup
  - Create n quorums.
  - Assign each gate to a quorum.
  - For each gate, create a multiplication triple via FHE.
- Online
  - Compute each gate over secret-shared values.
  - Reshare the result to parent gates.
  - Propagate the final result to all quorums.

InitTriple

Algorithm 2 InitTriple

Usage. Each party P_i jointly computes a triple (u_i, v_i, w_i), where $\langle u_1, ..., u_N \rangle_\tau$, $\langle v_1, ..., v_N \rangle_\tau$, and $\langle w_1, ..., w_N \rangle_\tau$ are sharings of u, v, and w respectively, where $u, v \in \mathbb{Z}_p$ are chosen uniformly at random, $w = u \cdot v$, and $\tau = (1/3 - \epsilon)N$.

<u>InitTriple()</u>:

1. For all $i \in [N]$, party P_i chooses values $a_i, b_i \in \mathbb{Z}_p$ uniformly at random, and broadcasts the pair $(\mathsf{Enc}(a_i), \mathsf{Enc}(b_i))$.
2. Let $\{(\mathcal{C}_{a_j}, \mathcal{C}_{b_j})\}_{j=1}^{N}$ be the set of pairs P_i receives from the previous step[9]. P_i computes

$$\mathcal{C}_u = \sum_{j=1}^{N} \mathcal{C}_{a_j},\ \mathcal{C}_v = \sum_{j=1}^{N} \mathcal{C}_{b_j}, \text{ and } \mathcal{C}_w = \mathcal{C}_u \cdot \mathcal{C}_v.$$

   Parties runs $\mathsf{CipherShare}(\mathcal{C}_u)$, $\mathsf{CipherShare}(\mathcal{C}_v)$, and $\mathsf{CipherShare}(\mathcal{C}_w)$ to generate three sharings $\langle \mathcal{C}_u \rangle_\tau$, $\langle \mathcal{C}_v \rangle_\tau$, and $\langle \mathcal{C}_w \rangle_\tau$.

3. For all $i \in [N]$, party P_i runs $u_i = \mathsf{DecPrivate}(\mathcal{C}_{u_i})$, $v_i = \mathsf{DecPrivate}(\mathcal{C}_{v_i})$, and $w_i = \mathsf{DecPrivate}(\mathcal{C}_{w_i})$.

Multiply

Algorithm 6 Multiply

Usage. Initially, parties jointly hold two sharings $\langle \alpha_1, ..., \alpha_N \rangle_\tau$ and $\langle \beta_1, ..., \beta_N \rangle_\tau$ of secret values $\alpha, \beta \in \mathbb{Z}_p$ respectively, where $\tau < (1/3 - \epsilon)N$. For $i \in [N]$, each party P_i also hold a triple (u_i, v_i, w_i) generated during the setup phase of the protocol. The algorithm computes a new sharing $\langle \gamma_1, ..., \gamma_N \rangle_\tau$ of $\gamma \in \mathbb{Z}_p$ such that $\gamma = \alpha \cdot \beta$.

<u>Multiply(α_i, β_i)</u>:

For all $i \in [N]$, party P_i computes $\varepsilon_i = \alpha_i + u_i$ and $\delta_i = \beta_i + v_i$ and runs Reconst(ε_i) and Reconst(δ_i) to learn ε and δ. Party P_i computes and returns $\gamma_i = w_i - \delta\alpha_i - \varepsilon\beta_i + \varepsilon\delta$.

Reshare

Algorithm 7 Reshare

Usage. Let Q be the quorum associated with gate G in circuit $\mathscr{C}$, and Q' be the quorum associated with a parent of G in $\mathscr{C}$. Initially, parties in Q hold a sharing $\langle \gamma_1, ..., \gamma_N \rangle_\tau$ of a secret $\gamma \in \mathbb{Z}_p$, where $\tau = (1/3 - \epsilon)N$. Using this algorithm, parties in Q jointly generate a *fresh* sharing of γ in Q'. More formally, they generate a new sharing $\langle \gamma'_1, ..., \gamma'_N \rangle_\tau$ of a value $\gamma' \in \mathbb{Z}_p$ in Q' such that $\gamma = \gamma'$.

Reshare(γ_i, Q'):

For all $i \in [N]$,

1. Party $P_i \in Q$ runs GenRand to jointly generate a sharing $\langle r_1, ..., r_N \rangle_{(\tau-1)}$ of a uniform random value $r \in \mathbb{Z}_p$.
2. $P_i \in Q$ computes $\gamma'_i = \gamma_i + i \cdot r_i$, and sends γ'_i to party $P_i \in Q'$.