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
- $< \frac{N}{9}$ 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

\[ \phi(x) \]

\[ C \]
Resharing

\[ \rho(x) \quad \phi(x) \]

Diagram with points and lines indicating relationships between \( \rho(x) \) and \( \phi(x) \).
Resharing

\[ \rho(x) \]

\[ \phi(x) \]

\[ x \cdot \rho(x) \]
Resharing

\[ \phi'(x) = \phi(x) + x \cdot \rho(x) \]
Multiplication over Shares

How to compute $a_i \cdot b_i$ from $a_i$ and $b_i$?
Beaver’s Multiplication Triples

\((u_i, v_i, w_i)\)

\[ w = u \cdot v \]

TFHE in Offline phase
Beaver’s Multiplication Triples

\[ c_i = w_i - \delta a_i - \varepsilon b_i + \varepsilon \delta \]
\[ \varepsilon_i = a_i + u_i \]
\[ \delta_i = b_i + v_i \]

In online phase
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

Number of Kilobytes sent per party per sorted element

Log number of Kilobytes sent vs Log number of parties
Microbenchmarks

Number of Kilobytes sent per party per sorted element

Log number of Kilobytes sent

Log number of parties

5 KB
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?
• One sorting \((n = 2^{25})\)

<table>
<thead>
<tr>
<th>Phase</th>
<th>% phase</th>
<th>% total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td></td>
<td>76%</td>
</tr>
<tr>
<td>Quorum building</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Key generation</td>
<td>95%</td>
<td></td>
</tr>
<tr>
<td>Triple generation</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>Online</td>
<td></td>
<td>24%</td>
</tr>
<tr>
<td>Circuit computation</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>Output propagation</td>
<td>99%*</td>
<td></td>
</tr>
</tbody>
</table>
## Costs Breakdown

<table>
<thead>
<tr>
<th>Phase</th>
<th>( n = 2^{10} )</th>
<th>( n = 2^{30} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>99%</td>
<td>8%</td>
</tr>
<tr>
<td>Online</td>
<td>1%</td>
<td>92%</td>
</tr>
</tbody>
</table>
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.
Algorithm 2  InitTriple

Usage. Each party $P_i$ jointly computes a triple $(u_i, v_i, w_i)$, where $\langle u_1, \ldots, u_N \rangle_\tau$, $\langle v_1, \ldots, v_N \rangle_\tau$, and $\langle w_1, \ldots, w_N \rangle_\tau$ 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 $\tau = (1/3 - \epsilon)N$.

InitTriple():

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 $(\text{Enc}(a_i), \text{Enc}(b_i))$.

2. Let $\{(C_{a_j}, C_{b_j})\}_{j=1}^N$ be the set of pairs $P_i$ receives from the previous step$^9$. $P_i$ computes

$$C_u = \sum_{j=1}^N C_{a_j}, \quad C_v = \sum_{j=1}^N C_{b_j}, \quad \text{and} \quad C_w = C_u \cdot C_v.$$ 

Parties run $\text{CipherShare}(C_u)$, $\text{CipherShare}(C_v)$, and $\text{CipherShare}(C_w)$ to generate three sharings $\langle C_u \rangle_\tau$, $\langle C_v \rangle_\tau$, and $\langle C_w \rangle_\tau$.

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


Algorithm 6  Multiply

Usage. Initially, parties jointly hold two sharings $\langle \alpha_1, \ldots, \alpha_N \rangle_\tau$ and $\langle \beta_1, \ldots, \beta_N \rangle_\tau$ 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 $\langle \gamma_1, \ldots, \gamma_N \rangle_\tau$ of $\gamma \in \mathbb{Z}_p$ such that $\gamma = \alpha \cdot \beta$.

Multiply($\alpha_i, \beta_i$):

For all $i \in [N]$, party $P_i$ computes $\varepsilon_i = \alpha_i + u_i$ and $\delta_i = \beta_i + v_i$ and runs $\text{Reconst}(\varepsilon_i)$ and $\text{Reconst}(\delta_i)$ to learn $\varepsilon$ and $\delta$. Party $P_i$ computes and returns $\gamma_i = w_i - \delta \alpha_i - \varepsilon \beta_i + \varepsilon \delta$. 
Algorithm 7  Reshare

Usage. Let $Q$ be the quorum associated with gate $G$ in circuit $\mathcal{C}$, and $Q'$ be the quorum associated with a parent of $G$ in $\mathcal{C}$. 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 $\gamma$ 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($\gamma_i, Q'$):

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 $\gamma'_i$ to party $P_i \in Q'$. 