Robust MPC: Asynchronous Responsiveness yet Synchronous Security

Chen-Da Liu-Zhang
ETH Zurich

Julian Loss
RUB

Ueli Maurer
ETH Zurich

Tal Moran
IDC Herzliya

Daniel Tschudi
Aarhus University
Multiparty Computation
Multiparty Computation
Multiparty Computation

**Correctness:** Output is correct

**Privacy:** Inputs remain private

**Termination:** Parties obtain output
Synchronous Model

Round structure:

• Each $P_i$ knows the current round
• Round $r$: $P_i$ reads round $r-1$ messages
  $P_i$ computes/sends round $r$ messages
• **Round $r$ messages are guaranteed to be delivered by round $r+1**
Synchronous Model

Round structure:

- Each $P_i$ knows the current round
- Round $r$: $P_i$ reads round $r-1$ messages
  $P_i$ computes/sends round $r$ messages
- Round $r$ messages are guaranteed to be delivered by round $r+1$

Can be achieved with synchronized clocks and channels with known delay upper bound $\Delta$
Synchronous Protocols

$t < \frac{n}{2}$

Input completeness
Synchronous Protocols

\[ t < \frac{n}{2} \]

Input completeness

\[ \Delta \gg \delta \], for network delay \( \delta \)

Time: \( T(\Delta) \)
Asynchronous Protocols

Greedy approach: $P_i$ continues as soon as it gets enough messages
Asynchronous Protocols

Greedy approach: $P_i$ continues as soon as it gets enough messages

Time: $T(\delta)$

$\delta$ network delay
Asynchronous Protocols

Greedy approach: $P_i$ continues as soon as it gets enough messages

- Time: $T(\delta)$
- $\delta$ network delay

- $t < \frac{n}{3}$
- Take into account $n - t$ inputs
<table>
<thead>
<tr>
<th></th>
<th>Correctness</th>
<th>Privacy</th>
<th>Termination</th>
<th>Number of Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong></td>
<td>$\frac{n}{2}$</td>
<td>$\frac{n}{2}$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td><strong>Asynchronous</strong></td>
<td>$\frac{n}{3}$</td>
<td>$\frac{n}{3}$</td>
<td>$\frac{n}{3}$</td>
<td>$\frac{2n}{3}$</td>
</tr>
</tbody>
</table>
# Synchronous vs Asynchronous

<table>
<thead>
<tr>
<th></th>
<th>Correctness</th>
<th>Privacy</th>
<th>Termination</th>
<th>Number of Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slow Output T(Δ)</td>
<td>Fast Output T(δ)</td>
</tr>
<tr>
<td>Synchronous</td>
<td>n/2</td>
<td>n/2</td>
<td>n</td>
<td>-</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>n/3</td>
<td>n/3</td>
<td>-</td>
<td>n/3</td>
</tr>
<tr>
<td>Our work</td>
<td>n/2</td>
<td>n/2</td>
<td>n</td>
<td>n/4</td>
</tr>
</tbody>
</table>

- Synchronous
  - Correctness: \( n/2 \)
  - Privacy: \( n/2 \)
  - Termination:
    - Slow Output: \( n \)
    - Fast Output: -
  - Number of Inputs: \( n \)

- Asynchronous
  - Correctness: \( n/3 \)
  - Privacy: \( n/3 \)
  - Termination:
    - Slow Output: -
    - Fast Output: \( n/3 \)
  - Number of Inputs: \( 2n/3 \)

- Our work
  - Correctness: \( n/2 \)
  - Privacy: \( n/2 \)
  - Termination:
    - Slow Output: \( n \)
    - Fast Output: \( n/4 \)
  - Number of Inputs: \( n \) or \( 3n/4 \)
Model

Global clock
[KMTZ13]

Network
unknown delay $\delta$

Time out

Corrupt up to $t$ parties arbitrarily
Schedule messages arbitrarily within $\delta$ clock ticks

Protocols can use $\Delta \gg \delta$
Synchronous over $\delta$-network

Each $P_i$ appends to each message its round number
Upon receiving all messages from round $r$:
  - Send messages for round $r+1$
  - Notify every party with round $r$
Upon receiving notifications from all parties for round $r$, continue with round $r+1$
Synchronous over $\delta$-network

Each $P_i$ appends to each message its round number
Upon receiving all messages from round $r$:
  • Send messages for round $r+1$
  • Notify every party with round $r$
Upon receiving notifications from all parties for round $r$, continue with round $r+1$

Problem: Corrupted $P_j$ does not send a message, parties need to wait $\Delta$ clock ticks to deduce $P_j$ is corrupted
Synchronous over $\delta$-network

Each $P_i$ appends to each message its round number
Upon receiving all messages from round $r$:
  • Send messages for round $r+1$
  • Notify every party with round $r$
Upon receiving notifications from all parties for round $r$, continue with round $r+1$

Problem: Corrupted $P_j$ does not send a message, parties need to wait $\Delta$ clock ticks to deduce $P_j$ is corrupted
Guarantees

Correctness and Privacy
\[\frac{n}{2}\]
Guarantees

Output guarantees depend on $n/2$
Guarantees

Output guarantees depend on $t \leq \frac{n}{4}$

Fast network & $t \leq \frac{n}{4}$

Correctness and Privacy $\frac{n}{2}$

$t \leq \frac{n}{4}$

$T_{\text{async}}(\delta)$
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Correctness and Privacy
\[ \frac{n}{2} \]
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Slow network or
\[ t > \frac{n}{4} \]

Correctness and Privacy
\[ \frac{n}{2} \]
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Slow network or
\[ t > \frac{n}{4} \]

Correctness and Privacy
\[ \frac{n}{2} \]
Guarantees

Output guarantees depend on

Correctness and Privacy $\frac{n}{2}$

Fast network & $t \leq \frac{n}{4}$

Slow network or $t > \frac{n}{4}$
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Slow network or
\[ t > \frac{n}{4} \]

Correctness and Privacy
\[ n/2 \]
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Slow network or
\[ t > \frac{n}{4} \]

Correctness and Privacy
\[ \frac{n}{2} \]
Guarantees

Output guarantees depend on

Correctness and Privacy $n/2$

Fast network & $t \leq \frac{n}{4}$

Slow network or $t > \frac{n}{4}$
Guarantees

Output guarantees depend on

Fast network &
\[ t \leq \frac{n}{4} \]

Slow network or
\[ t > \frac{n}{4} \]

Correctness and Privacy
\[ \frac{n}{2} \]
Road map

$\Pi_{ASYNCE}$
Correctness/Privacy: $\frac{n}{2}$
Fast Output: $\frac{n}{4}$
Road map

\[ \Pi_{ASYNC} \]

- Correctness/Privacy: \( \frac{n}{2} \)
- Fast Output: \( \frac{n}{4} \)

Modify the protocol by [Coh16], using BA with increased validity and consistency, but lower termination [LosMor17]
Modify the protocol by [Coh16], using BA with increased validity and consistency, but lower termination [LosMor17]
Road map

$\Pi_{SYNC}$
Correctness/Privacy: $\frac{n}{2}$
Slow Output: $n$

$\Pi_{ASYNC}$
Correctness/Privacy: $\frac{n}{2}$
Fast Output: $\frac{n}{4}$

Compiler

$\Pi$
Correctness/Privacy: $\frac{n}{2}$
Fast Output: $\frac{n}{4}$
Slow Output: $n$

Modify the protocol by [Coh16], using BA with increased validity and consistency, but lower termination [LosMor17]
Compiler

\[ \Pi_{ASYNC} \xrightarrow{\text{yasync}} \]
Use an asynchronous protocol and a synchronous protocol as fallback*

*[PasShi17, LosMor17, GPS19]*
Compiler

Use an asynchronous protocol and a synchronous protocol as fallback*

\[ \Pi_{ASYNC} \xrightarrow{\text{async}} \Pi_{SYNC} \xrightarrow{\text{sync}} \]

Problem:
Adversary can learn two outputs, $y_{async}$ and $y_{sync}$!

* [PasShi17, LosMor17, GPS19]
Compiler

Asynchronous Phase

Synchronous Phase

Time out
Compiler

Asynchronous Phase

Synchronous Phase

Time out

\( y_{async} \)
Compiler

Asynchronous Phase

Synchronous Phase

Time out

\( \gamma_{async} \)
Compiler

Asynchronous Phase

Synchronous Phase

Time out

$y_{sync}$
Compiler

Asynchronous Phase

Synchronous Phase

Time out
Compiler

Setup: Threshold encryption and digital signatures

\[ \Pi_{ASYNC} \rightarrow [y_{async}] \rightarrow Decrypt \]

\( \frac{3}{4} n \) decryption shares to reconstruct

Synchronous Phase

Time out
Compiler

Setup: Threshold encryption and digital signatures

\[ \Pi_{ASYNC} \rightarrow \text{Decrypt} \]

\( \frac{3}{4} n \) decryption shares to reconstruct

Time out

Synchronous Phase
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

Synchronous Phase

Time out
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

Clock

Time out

Check \( \Pi_{SYNC} \)
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

\( \Pi_{SYNC} \)

\( \gamma_{async} \)

Time out
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

\[ \Pi \]

\[ \text{Check} \]

\[ \Pi_{SYNC} \]

Time out
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

\[ \text{Check} \quad \Pi_{SYNC} \]

\( \gamma_{async} \)

Time out
Compiler

Setup: Threshold encryption and digital signatures

Asynchronous Phase

\[
\Pi \quad \text{SYNC}
\]

Check \quad \nabla \quad \Pi_{\text{SYNC}}

\text{Time out}

\text{\$sync}
Compiler

Setup: Threshold encryption and digital signatures

\[\Pi_{ASYNC} \rightarrow [yasync] \rightarrow Decrypt \rightarrow Check \rightarrow \bot \rightarrow \Pi_{SYNC} \rightarrow y_{sync}\]

Time out
Compiler

Setup: Threshold encryption and digital signatures

\[ \Pi_{ASYNC} \rightarrow [y_{async}] \rightarrow Decrypt \rightarrow y_{async} \]

\[ \Pi_{SYNC} \rightarrow \bot \rightarrow Check \rightarrow y_{sync} \]

Time out
1. Sign \([y_{async}]\) and send it to each \(P_j\).
2. Once collected list \(L\) of \(\frac{3}{4}n\) signatures on \(c\), send decryption share \(d_i = \text{Dec}_{d_{ki}}(c)\).
3. Once received \(\frac{3}{4}n\) shares, reconstruct the output \(y_{async}\).
1. Sign $[y_{async}]$ and send it to each $P_j$

2. Once collected list $L$ of $\frac{3}{4}n$ signatures on $c$, send decryption share $d_i = \text{Dec}_{d_{ki}}(c)$

3. Once received $\frac{3}{4}n$ shares, reconstruct the output $y_{async}$

>: Synchronously BC $(L,c)$, if received from Step 2.

After BC: If any $(L,c)$ received, send $d_i$ and reconstruct $y_{async}$

Otherwise, output $\bot$
1. Sign \( [y_{async}] \) and send it to each \( P_j \)
2. Once collected list \( L \) of \( \frac{3}{4} n \) signatures on \( c \), send decryption share \( d_i = Dec_{d_{ki}}(c) \)
3. Once received \( \frac{3}{4} n \) shares, reconstruct the output \( y_{async} \)

\( \otimes \): Synchronously BC \((L,c)\), if received from Step 2.

After BC: If any \((L,c)\) received, send \( d_i \) and reconstruct \( y_{async} \)
Otherwise, output \( \bot \)
Guarantees

Output guarantees depend on

Fast network & \( t \leq \frac{n}{4} \)

Slow network or \( t > \frac{n}{4} \)
Guarantees

Output guarantees depend on

Fast network & 
\[ t \leq \frac{n}{4} \]

Slow network or 
\[ t > \frac{n}{4} \]
References and Credits

ETH Zurich, Department of Computer Science
lichen@inf.ethz.ch

References:


[GPS19]: Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. CRYPTO 2019.

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