Fluid MPC: Secure Multiparty Computation with Dynamic Participants

Arka Rai Choudhuri      Aarushi Goel      Matthew Green
Abhishek Jain      Gabriel Kaptchuk
Secure Multiparty Computation

\[ x_1, x_2, x_3, x_4, x_5 \]
Secure Multiparty Computation

\[ y = f(x_1, x_2, x_3, x_4, x_5) \]
Secure Multiparty Computation

\[ y = f(x_1, x_2, x_3, x_4, x_5) \]

Adversary learns the same amount of information in the two scenarios
Efficient MPC and Emerging Applications

MPC protocols are becoming increasingly efficient. They can be used to compute complex functionalities such as:

- Training machine learning algorithms on massive, distributed datasets.
- Simulating large RAM programs on distributed datasets.
- Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.

The circuit representations of these computations could be extremely deep.
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.

• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
  • Simulating large RAM programs on distributed datasets
  • Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.

• The circuit representations of these computations could be extremely deep.
Efficient MPC and Emerging Applications

- MPC protocols are becoming increasingly efficient.
- Can be used to compute complex functionalities such as:
  - Training machine learning algorithms on massive, distributed datasets.
  - Simulating large RAM programs on distributed datasets.
  - Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.
  - The circuit representations of these computations could be extremely deep.
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.

• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.
• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
  • Simulating large RAM programs on distributed datasets
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.
• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
  • Simulating large RAM programs on distributed datasets
  • Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.

• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
  • Simulating large RAM programs on distributed datasets
  • Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.

• The circuit representations of these computations could be extremely deep.
Efficient MPC and Emerging Applications

• MPC protocols are becoming increasingly efficient.

• Can be used to compute complex functionalities such as:
  • Training machine learning algorithms on massive, distributed datasets.
  • Simulating large RAM programs on distributed datasets
  • Optimization programs over a complex, high dimensional space, where the constraints of the dimensions are held by different players.

• The circuit representations of these computations could be extremely deep.

**Issue:** Evaluating these functionalities could take up to several hours or even days.
Prior Work: Static MPC
Prior Work: Static MPC

Round 1
Prior Work: Static MPC

Round 1

Round 2
Prior Work: Static MPC

Round 1  |  Round 2  |  Round 3
Prior Work: Static MPC

Round 1  Round 2  Round 3  ...  ...
Prior Work: Static MPC

Round 1  Round 2  Round 3  ...

I need to leave
Prior Work: Static MPC

Round 1 | Round 2 | Round 3 | ... | ...

Dropped out due to connectivity issue
Prior Work: Static MPC

Requiring all participants to stay online throughout the computation is an unrealistic expectation.
Prior Work: Static MPC

Can we design MPC protocols with Dynamic Participants?
MPC with Dynamic Participants
MPC with Dynamic Participants

A group of parties start the computation
MPC with Dynamic Participants

After some time two parties have to leave.
MPC with Dynamic Participants

And a new party wants to join the computation
MPC with Dynamic Participants

The previous group of parties securely distributes information about the computation so far, to the new group.
MPC with Dynamic Participants

Given this information, the new group continues with the rest of the computation.
MPC with Dynamic Participants

Again, after some time, a party has to leave
MPC with Dynamic Participants

And an old party wants to re-join the computation
MPC with Dynamic Participants

This group will again securely distribute information about the computation thus far, with the new group of parties.
MPC with Dynamic Participants

This group will continue with the rest of the computation
MPC with Dynamic Participants

This reduces the burden of computation on individual parties
MPC with Dynamic Participants

This reduces the burden of computation on individual parties.

Parties with low computational resources can also participate for a small time.
MPC with Dynamic Participants

This reduces the burden of computation on individual parties.

While parties with more time and computational resources can help with the computation for a longer time.
MPC with Dynamic Participants

This will result in a weighted, privacy preserving distributed computing system.
MPC as a Service

- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants
MPC as a Service

- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants

Dynamic Peer-to-peer networks.
MPC as a Service

- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants

Dynamic Peer-to-peer networks.
- Powered by volunteer nodes - that can come and go as they wish.
- Very Successful!
MPC as a Service

- Powered by volunteer nodes that can come and go as they wish.
- Very Successful!

**MPC with Dynamic Participants**
- Allows Participants to join and leave at will
- Reduces burden of computation on individual participants

**Dynamic Peer-to-peer networks.**
- Powered by volunteer nodes - that can come and go as they wish.
- Very Successful!

Compatible with each other
MPC as a Service

MPC with Dynamic Participants + Dynamic Peer-to-peer networks.
MPC as a Service

MPC with Dynamic Participants + Dynamic Peer-to-peer networks.

Volunteer networks capable of private computation.
MPC as a Service

MPC with Dynamic Participants + Dynamic Peer-to-peer networks.

Volunteer networks capable of private computation.

MPC-as-a-service framework - anyone can volunteer to participate irrespective of their computational power or availability.

Clients can delegate computations to such services.
Player Replaceability

- Byzantine Agreement [Mic17, CM19]: After every round, the current set of players can be replaced by new ones.
Player Replaceability

- **Byzantine Agreement [Mic17, CM19]**: After every round, the current set of players can be replaced by new ones.

- **Blockchains [GHMVZ17]**: This idea is used in the design of Algorand.
  - Helps mitigate targeted attacks on chosen participants after their identity is revealed.
Related Work

• Proactive MPC [OY91]
  • Static participants
  • Mobile adversaries
Related Work

• **Proactive MPC [OY91]**
  • Static participants
  • Mobile adversaries

• **Secret Sharing with dynamic participants [GKMPS20, BGGHKLLR20]**
  • Computational setting
  • Guaranteed output delivery
Our Contributions

Fluid MPC: A formal model for MPC with dynamic participants
Our Contributions

Fluid MPC: A formal model for MPC with dynamic participants

Fluidity is the minimum commitment a party needs to make for participating in the protocol.
Our Contributions

Fluid MPC: A formal model for MPC with dynamic participants

Semi-Honest BGW protocol can be adapted to the Fluid MPC setting, where each party is required to speak only in one round (max fluidity)

Fluidity is the minimum commitment a party needs to make for participating in the protocol.
Our Contributions

Fluid MPC: A formal model for MPC with dynamic participants

Semi-Honest BGW protocol can be adapted to the Fluid MPC setting, where each party is required to speak only in one round (max fluidity)

A compiler that transforms “certain” semi-honest Fluid MPC protocols into maliciously secure protocols:

- security with abort
- $2 \times$ communication complexity
- Preserves fluidity

Fluidity is the minimum commitment a party needs to make for participating in the protocol.
Our Contributions

Fluid MPC: A formal model for MPC with dynamic participants

Semi-Honest BGW protocol can be adapted to the Fluid MPC setting, where each party is required to speak only in one round (max fluidity)

A compiler that transforms “certain” semi-honest Fluid MPC protocols into maliciously secure protocols:

- security with abort
- $2 \times$ communication complexity
- Preserves fluidity

Implementation of our maliciously secure protocol based on BGW

Fluidity is the minimum commitment a party needs to make for participating in the protocol.
Fluid MPC Model
Modeling Dynamic Computation

• **Client-server** model

• Clients delegate computation to volunteer servers
Modeling Dynamic Computation

• **Client-server** model
• Clients delegate computation to volunteer servers

Input Stage

Clients pre-process their inputs and hand them to the servers
Modeling Dynamic Computation

• **Client-server** model

• Clients delegate computation to volunteer servers

**Input Stage**

Clients pre-process their inputs and hand them to the servers

**Execution Stage**

*Dynamic servers* participate to compute the function
Modeling Dynamic Computation

• Client-server model
• Clients delegate computation to volunteer servers

Input Stage
Clients pre-process their inputs and hand them to the servers

Execution Stage
*Dynamic servers* participate to compute the function

Output Stage
Clients reconstruct the output of the function
Modeling Execution Stage
Modeling Execution Stage

Epoch $i$  Epoch $i+1$  Epoch $i+2$
Modeling Execution Stage

Epoch $i$  Epoch $i+1$  Epoch $i+2$
Modeling Execution Stage

Epoch $i$  
Compute Phase  
Hand-off Phase

Epoch $i+1$  
Compute Phase  
Hand-off Phase

Epoch $i+2$  
Compute Phase  
Hand-off Phase
Modeling Execution Stage

Epoch $i$  
Compute Phase  
Hand-off Phase  
Epoch $i+1$  
Compute Phase  
Hand-off Phase  
Epoch $i+2$  
Compute Phase  
Hand-off Phase

Committee $S^i$
Modeling Execution Stage

Compute Phase

Epoch $i$

Hand-off Phase

Compute Phase

Epoch $i+1$

Hand-off Phase

Compute Phase

Epoch $i+2$

Hand-off Phase

Committee $S^i$

Committee $S^{i+1}$
Modeling Execution Stage

Epoch $i$  
Compute Phase  
Hand-off Phase

Epoch $i+1$  
Compute Phase  
Hand-off Phase

Epoch $i+2$  
Compute Phase  
Hand-off Phase

Committee $S^i$  
Committee $S^{i+1}$
Modeling Execution Stage

 Epoch $i$
 Compute Phase
 Hand-off Phase
 Compute Phase
 Hand-off Phase
 Compute Phase
 Hand-off Phase

 Committee $S^i$

 Committee $S^{i+1}$

 Committee $S^{i+2}$
Corruption Threshold

- **Clients**: Honest Majority or Dishonest majority
- **Servers**: Honest Majority or Dishonest majority
Corruption Threshold

- **Clients**: Honest Majority or Dishonest majority
- **Servers**: Honest Majority or Dishonest majority

**Our Choice**

- Honest majority of clients
- Honest majority of servers in each committee
Fluid MPC Protocol

Committee Selection/Corruption + Protocol Execution given these Committees
Fluid MPC Protocol

Committee Selection/Corruption

Protocol Execution given these Committees
Fluid MPC Protocol

Committee Selection/Corruption

- Committee Formation
- Committee Corruption
- Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
Fluid MPC Protocol

Committee Selection/Corruption
- Committee Formation
- Committee Corruption
- Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
Committees: When are they formed?

**Static Committee Formation:** Committee for each epoch is known at the start of the protocol or the execution stage.
Committees: When are they formed?

Static Committee Formation: Committee for each epoch is known at the start of the protocol.
Committees: When are they formed?

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Committee</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1" alt="Committee Members" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image2" alt="Committee Members" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image3" alt="Committee Members" /></td>
</tr>
</tbody>
</table>

**Static Committee Formation:** Committee for each epoch is known at the start of the protocol.
Committees: When are they formed?

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Committee</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Static Committee Formation:** Committee for each epoch is known at the start of the protocol.

Too Restrictive!
Committees: When are they formed?

On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.
Committees: When are they formed?

On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.
Committees: When are they formed?

On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.
Committees: When are they formed?

On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.
Committees: When are they formed?

On-the-fly Committee Formation: Committee for each epoch is known at the start of the hand-off phase of the previous epoch.
Committees: How are they formed?

On-the-fly Committee Formation:

Volunteer: Anyone who volunteers can join the computation (Corruption threshold is difficult to enforce)
Committees: How are they formed?

On-the-fly Committee Formation:

Volunteer: Anyone who volunteers can join the computation (Corruption threshold is difficult to enforce)

Elected: Anyone can nominate themselves and an election process decides which nominees will participate (e.g., [BGGHKLRR20, GHMNY20] uses proof-of-stake blockchains)
Fluid MPC Protocol

Committee Selection/Corruption
  - Committee Formation
  - Committee Corruption
  - Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
Committee Corruption

When can a server be corrupted?

Corrupted at the time of committee formation

Static Corruption
Committee Corruption

When can a server be corrupted?

Adaptive Corruption
Committee Corruption

When can a server be corrupted?

Adaptive Corruption

Hand-off Phase Compute Phase Hand-off Phase
Committee Corruption

When can a server be corrupted?

Adaptive Corruption
Committee Corruption

When can a server be corrupted?

Adaptive Corruption

Corrupted at any time

Hand-off Phase  Compute Phase  Hand-off Phase
Committee Corruption

**When can a server be corrupted?**

Adaptive Corruption

Our Choice

Hand-off Phase | Compute Phase | Hand-off Phase

Corrupted at any time
Fluid MPC Protocol

Committee Selection/Corruption

- Committee Formation
- Committee Corruption

Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?
What effect does corrupting a server have on the prior epochs where it participated?

Can be prevented by only allowing disjoint committees
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?

Can be prevented by only allowing disjoint committees

If there is overlap across committees, a server can only be corrupted if it does not violate the corruption threshold of prior epochs.
Effect of Committee Corruption on Prior Epochs

What effect does corrupting a server have on the prior epochs where it participated?

Can be prevented by only allowing disjoint committees

If there is overlap across committees, a server can only be corrupted if it does not violate the corruption threshold of prior epochs. Similar to being passively corrupted in prior epochs.
Fluid MPC Protocol

Committee Selection/Corruption

Committee Formation

Committee Corruption

Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
Fluid MPC Protocol

Committee Selection/Corruption
- Committee Formation
- Committee Corruption
- Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
- Requirements/Challenges
  - Semi-Honest
  - Malicious
Fluid MPC Protocol

Committee Selection/Corruption
- Committee Formation
- Committee Corruption
- Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees
- Requirements/Challenges
  - Semi-Honest
  - Malicious
Requirements: Small State Complexity

Since states need to be transferred after every epoch, state complexity has a direct effect on communication complexity.
Requirements: Small State Complexity

Since states need to be transferred after every epoch, state complexity has a direct effect on communication complexity.

State size of each party should be independent of the depth of the circuit.
Fluidity is the minimum commitment a server needs to make for participating in the protocol.

Measured by the number of rounds in an epoch.
Maximal Fluidity

Our Choice

Essentially, each party is only required to communicate in one round

Committee $S^i$

Committee $S^{i+1}$

Silent compute phase

1 Round of unidirectional hand-off phase

1 round epoch
Requirements: Secure State Transfer

Hand-off Phase
Requirements: Secure State Transfer
Requirements: Secure State Transfer

Hand-off Phase

Adv learns the private state of 2 out of 3 parties in the first committee
Requirements: Secure State Transfer

This naïve way handing-off states between committees in a one-to-one manner could break privacy.

Need a secure state transferring mechanism
Requirements: Checklist

- Max Fluidity
- Small State Size
- Secure State Transfer
Fluid MPC Protocol

Committee Selection/Corruption

- Committee Formation
- Committee Corruption
- Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees

- Requirements/Challenges
- Semi-Honest
- Malicious
Semi-honest BGW
Semi-honest BGW

Gate-by-Gate evaluation on secret shared inputs
Semi-honest BGW

Input sharing: $t$-out-of-$n$ shares of inputs
Semi-honest BGW

Input sharing: \( t \)-out-of-\( n \) shares of inputs
Semi-honest BGW

Gate-by-Gate Evaluation

Compute \( [e]_{2t} = [a]_t \times [b]_t \)

Compute \( [f]_t = [c]_t + [d]_t \)

Input sharing: \( t \)-out-of-\( n \) shares of inputs
Semi-honest BGW

Input sharing: $t$-out-of-$n$ shares of inputs

Gate-by-Gate Evaluation

Compute

\[
[e]_{2t} = [a]_t \times [b]_t
\]

\[
[e]_{2t} \leftarrow [e]_{2t}
\]
Semi-honest BGW

Gate-by-Gate Evaluation

Compute

\[
[e]_{2t} = [a]_t \times [b]_t
\]

\[
[[e]_{2t}]_t \leftarrow [e]_{2t}
\]

Exchange

[[e]_{2t}]_t

(Shares of Shares)

Input sharing: \( t \)-out-of-\( n \) shares of inputs
Semi-honest BGW

Gate-by-Gate Evaluation

Compute \([e]_{2t} = [a]_t \times [b]_t\)

Exchange (Shares of Shares)

Compute \([e]_t \leftarrow [e]_{2t}\)

Input sharing: \(t\)-out-of-\(n\) shares of inputs
Semi-honest BGW

Gate-by-Gate Evaluation

Compute \([e]_{2t} = [a]_t \times [b]_t\)

\([e]_{2t} \leftarrow [e]_{2t}\)

Exchange (Shares of Shares)

Compute \([e]_t \leftarrow [[e]_{2t}]_t\)

Output Reconstruction

Compute \([f]_t = [c]_t + [d]_t\)

Input sharing: \(t\text{-out-of-}n\) shares of inputs
Semi-honest Fluid-BGW
Semi-honest Fluid-BGW

Input Phase: Clients send $t$-out-of-$n$ shares of inputs to the first committee
Semi-honest Fluid-BGW

Execution Stage

Layer-wise computations
Committee \( i \) computes layer \( i \)

Input Phase: Clients send \( t \)-out-of-\( n \) shares of inputs to the first committee
Semi-honest **Fluid-BGW**

**Execution Stage**

**Computation Phase:** \([e]_t \leftarrow [[e]_{2t}]_t\)

**Handoff Phase:** \([[[e]_{2t}]_t]_t\)

**Computation Phase:** \([e]_{2t} = [a]_t \times [b]_t\)  
\([[[e]_{2t}]_t]_t \leftarrow [e]_{2t}\)

**Computation Phase:** \([f]_t \leftarrow [[[f]_t]_t]_t\)

**Handoff Phase:** \([[f]_t]_t\)

**Computation Phase:** \([f]_{2t} = [c]_t + [d]_t\)  
\([[f]_t]_t \leftarrow [f]_t\)

**Input Phase:** Clients send \(t\)-out-of-\(n\) shares of inputs to the first committee
Semi-honest Fluid-BGW

**Input Phase:** Clients send $t$-out-of-$n$ shares of inputs to the first committee.

- **Execution Stage**
  - **Computation Phase:** $[e]_t \leftarrow [[e]_{2t}]_t$
  - **Handoff Phase:** $[[e]_{2t}]_t$

- **Computation Phase:** $[e]_{2t} = [a]_t \times [b]_t$
  - **Handoff Phase:** $[[e]_{2t}]_t \leftarrow [e]_{2t}$

- **Computation Phase:** $[f]_t \leftarrow [[f]_t]_t$
  - **Handoff Phase:** $[[f]_t]_t$

- **Computation Phase:** $[f]_t = [c]_t + [d]_t$
  - **Handoff Phase:** $[[f]_t]_t \leftarrow [f]_t$
Semi-honest Fluid-BGW

Input Phase: Clients send $t$-out-of-$n$ shares of inputs to the first committee

Execution Stage

Computation Phase: $[e]_t \leftarrow [[e]_{2t}]_t$
of Epoch 2

Handoff Phase: $[[e]_{2t}]_t$

Computation Phase: $[e]_{2t} = [a]_t \times [b]_t$
$[[e]_{2t}]_t \leftarrow [e]_{2t}$
of Epoch 1

Handoff Phase: $[[f]_t]_t$

Computation Phase: $[f]_t = [c]_t + [d]_t$
$[[f]_t]_t \leftarrow [f]_t$
of Epoch 1

Computation Phase: $[f]_t \leftarrow [[f]_t]_t$
of Epoch 2
Semi-honest Fluid-BGW

**Execution Stage**

**Computation Phase:** $[e]_t \leftarrow [[e]_{2t}]_t$
- of Epoch 2

**Handoff Phase:** $[[e]_{2t}]_t$

**Computation Phase:**
- $[e]_{2t} = [a]_t \times [b]_t$
- $[[e]_{2t}]_t \leftarrow [e]_{2t}$
- of Epoch 1

**Handoff Phase:**
- $[[f]_{2t}]_t$

**Computation Phase:**
- $[f]_t \leftarrow [[f]_t]_t$
- of Epoch 2

**Computation Phase:**
- $[f]_t = [c]_t + [d]_t$
- $[[f]_t]_t \leftarrow [f]_t$
- of Epoch 1

**Output Phase:**
- $g \leftarrow [g]_t$

**Input Phase:** Clients send $t$-out-of-$n$ shares of inputs to the first committee.
Semi-honest Fluid-BGW

**Execution Stage**

**Computation Phase:** $[e]_t \leftarrow [[e]_{2t}]_t$

**Handoff Phase:** $[[e]_{2t}]_t$

**Computation Phase:** $[e]_{2t} = [a]_t \times [b]$

$[[e]_{2t}]_t \leftarrow [e]_{2t}$

**Output Phase**

$a \leftarrow [a]_t$

**Computation Phase:** $[f]_t \leftarrow [[f]_t]_t$

**Handoff Phase:** $[[f]_t]_t$

**Computation Phase:** $[f]_t = [c]_t + [d]_t$

$[[f]_t]_t \leftarrow [f]_t$

**Input Phase:** Clients send $t$-out-of-$n$ shares of inputs to the first committee

- Max Fluidity
- Small State Size
- Secure State Transfer

**Transfer**
Fluid MPC Protocol

Committee Selection/Corruption

Committee Formation

Committee Corruption

Effect of Committee Corruption on Prior Epochs

Protocol Execution given these Committees

Requirements/Challenges

Semi-Honest

Malicious
Shortcomings of Natural Solutions

Need to Verify Honest Behavior

Implementing a gate-by-gate check
Requires more interaction

- Max Fluidity
- Small State Size
- Secure State Transfer
Shortcomings of Natural Solutions

Need to Verify Honest Behavior

Implementing a gate-by-gate check
Requires more interaction

- Max Fluidity
- Small State Size
- Secure State Transfer

Using NIZKs
May require access to all prior rounds

- Max Fluidity
- Small State Size
- Secure State Transfer
Additive Attack Paradigm [GIPST14]
Additive Attack Paradigm [GIPST14]

Most secret sharing based semi-honest protocols are secure against malicious adversaries up to additive attacks:
Most secret sharing based semi-honest protocols are secure against malicious adversaries up to additive attacks:

\[ X + \] \[ a + \varepsilon_a \] \[ b + \varepsilon_b \] 
\[ c + \varepsilon_c \] \[ d + \varepsilon_d \] 

\[ g + \varepsilon_g \] 
\[ e + \varepsilon_e \] \[ f + \varepsilon_f \] 

\[ g \] 
\[ e \] \[ f \] 
\[ a \] \[ b \] \[ c \] \[ d \] 

\[ + \] 
\[ X \] 
\[ X \] 

\[ \varepsilon \] additive errors are independent of the actual wire values
Modern efficient maliciously secure protocols rely on this additive attack paradigm.

Semi-honest execution
Modern efficient maliciously secure protocols rely on this additive attack paradigm.

Semi-honest execution

\[ X + X \]

Check validity of all the MACs at the end

\[ MAC(r, g) \]

\[ MAC(r, a) \quad MAC(r, b) \quad MAC(r, c) \quad MAC(r, d) \]
Modern efficient maliciously secure protocols rely on this additive attack paradigm.

Check validity of all the MACs at the end.
Efficient Maliciously Secure Protocols [DPSZ12, CGHIKLN18]

\[ g = ef + \epsilon_g \]
\[ r_g = (re)f + \epsilon_g' \]
\[ e = ab + \epsilon_e \]
\[ f = c + d + \epsilon_f \]
\[ re = (ra)b + \epsilon_e' \]
\[ a + \epsilon_a \quad b + \epsilon_b \]
\[ c + \epsilon_c \quad d + \epsilon_d \]
\[ ra + \epsilon_a' \quad rb + \epsilon_b' \]
\[ rc + \epsilon_c' \quad rd + \epsilon_d' \]
Efficient Maliciously Secure Protocols [DPSZ12,CGHIKLN18]

For each gate, check if:

\[ r \ (e) \ =?\ = \ re \]
Efficient Maliciously Secure Protocols [DPSZ12,CGHIKLN18]

For each gate, check if:

\[ r (e) =?= re \]
\[ r (ab + ε_e) =?= (ra)b + ε'_e \]
Efficient Maliciously Secure Protocols [DPSZ12, CGHIKLN18]

\[ e = ab + \varepsilon_e \]
\[ f = c + d + \varepsilon_f \]
\[ g = ef + \varepsilon_g \]

\[ r = (ra)b + \varepsilon_e' \]
\[ g = (re)f + \varepsilon_g' \]

\[ r = rc + rd + \varepsilon_f' \]

Sample a random \( \alpha_k \) for each gate

Consolidated check:

\[ r \sum_{k \in |C|} \alpha_k(z_k) = ? = \sum_{k \in |C|} \alpha_k(rz_k) \]
Maliciously secure Fluid MPC

Semi-honest Fluid BGW  Additive Attack Paradigm?  Maliciously secure Fluid MPC
Maliciously secure Fluid MPC

Additive Attack Paradigm?

Semi-honest Fluid BGW \rightarrow Maliciously secure Fluid MPC

We want this transformation to preserve the communication complexity and fluidity of fluid BGW.
Maliciously secure Fluid MPC

Additive Attack Paradigm?

We want this transformation to preserve the communication complexity and fluidity of fluid BGW

Observation: Additive Attack Paradigm extends to the Fluid MPC setting in a natural way
Maliciously secure Fluid MPC

Can we use known techniques in the additive attack paradigm?
Maliciously secure Fluid MPC

Can we use known techniques in the additive attack paradigm?

If the linear combination is computed at the end
the values of \( rz \) and \( z \) must have been passed along
as part of the state till the end of the protocol.
Maliciously secure Fluid MPC

Can we use known techniques in the additive attack paradigm?

If the linear combination is computed at the end, the values of \( rz \) and \( z \) must have been passed along as part of the state till the end of the protocol.

If the linear combination is computed incrementally layer-by-layer, the \( \alpha \) values will have to be generated on the fly, which may take many rounds.

- [ ] Max Fluidity
- [ ] Small State Size
- [✓] Secure State Transfer

- [ ] Max Fluidity
- [✓] Small State Size
- [✓] Secure State Transfer
Maliciously secure Fluid MPC: Our Idea

\[ x_1 \ x_2 \quad x_3 \ x_4 \quad x_5 \ x_6 \quad \ldots \quad x_{2w-1} \ x_{2w} \]
Maliciously secure Fluid MPC: Our Idea

\[\begin{align*}
\alpha_1 & \times x_1 \\
\alpha_2 & \times x_2 \\
\alpha_3 & \times x_3 \\
\alpha_w & \times x_w \\
\end{align*}\]
Maliciously secure Fluid MPC: Our Idea
Maliciously secure Fluid MPC: Our Idea

\[ \alpha \cdot x \]

\[ \beta \cdot z \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_{2w-1} \quad x_{2w} \]

\[ \beta^2 \quad \alpha_1 \beta^1 \quad \alpha_2 \beta^1 \quad \alpha_3 \beta^1 \quad \alpha_w \beta^1 \]

\[ \beta^1 \quad \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_w \]
Maliciously secure Fluid MPC: Our Idea

\[
\begin{align*}
&x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_{2w-1} \ x_{2w} \\
&z_1^1 \ z_2^1 \ z_3^1 \ z_4^1 \ \cdots \ z_w^1 \\
&z_1^2 \ z_2^2 \ z_3^2 \ z_4^2 \ \cdots \ z_w^2 \\
&\beta^3 \ \alpha_1 \beta^2 \ \alpha_2 \beta^2 \ \alpha_3 \beta^2 \ \alpha_w \beta^2 \\
&\beta^2 \ \alpha_1 \beta^1 \ \alpha_2 \beta^1 \ \alpha_3 \beta^1 \ \alpha_w \beta^1 \\
&u_0 = 0 \\
&\beta^1 \ \alpha_1 \ \alpha_2 \ \alpha_3 \ \alpha_w
\end{align*}
\]
Maliciously secure Fluid MPC: Our Idea

\[
\begin{align*}
\beta^3 & \quad \alpha_1 \beta^2 & \quad \alpha_2 \beta^2 & \quad \alpha_3 \beta^2 & \quad \alpha_w \beta^2 \\
u_1 &= u_0 + \alpha_1 \beta^1 z_1^1 + \alpha_2 \beta^1 z_2^1 + \cdots + \alpha_w \beta^1 z_w^1 \\
\beta^2 & \quad \alpha_1 \beta^1 & \quad \alpha_2 \beta^1 & \quad \alpha_3 \beta^1 & \quad \alpha_w \beta^1 \\
u_0 &= 0 \\
\beta^1 & \quad \alpha_1 & \quad \alpha_2 & \quad \alpha_3 & \quad \alpha_w
\end{align*}
\]
Maliciously secure Fluid MPC: Our Idea

\[
\begin{align*}
\beta^3 &\quad \alpha_1\beta^2 &\quad \alpha_2\beta^2 &\quad \alpha_3\beta^2 &\quad \alpha_w\beta^2 \\
\beta^2 &\quad \alpha_1\beta^1 &\quad \alpha_2\beta^1 &\quad \alpha_3\beta^1 &\quad \alpha_w\beta^1 \\
\beta^1 &\quad \alpha_1 &\quad \alpha_2 &\quad \alpha_3 &\quad \alpha_w
\end{align*}
\]

\[u_1 = u_0 + \alpha_1\beta^1 z_1^1 + \alpha_2\beta^1 z_2^1 + \ldots + \alpha_w\beta^1 z_w^1\]

\[u_0 = 0\]
Maliciously secure Fluid MPC: Our Idea

\[
\begin{align*}
\beta^3 &\quad \alpha_1 \beta^2 &\quad \alpha_2 \beta^2 &\quad \alpha_3 \beta^2 &\quad \alpha_w \beta^2 \\
\beta^2 &\quad \alpha_1 \beta^1 &\quad \alpha_2 \beta^1 &\quad \alpha_3 \beta^1 &\quad \alpha_w \beta^1 \\
\beta^1 &\quad \alpha_1 &\quad \alpha_2 &\quad \alpha_3 &\quad \alpha_w
\end{align*}
\]

\[
u_1 = u_0 + \alpha_1 \beta^1 z_1 + \alpha_2 \beta^1 z_2 + \cdots + \alpha_w \beta^1 z_w
\]

\[
u_0 = 0
\]
Maliciously secure Fluid MPC: Our Idea

\[ u_1 = u_0 + \alpha_1 \beta^1 z_1^1 + \alpha_2 \beta^1 z_2^1 + \ldots + \alpha_w \beta^1 z_w^1 \]

\[ u_0 = 0 \]

\[ \beta^3 \alpha_1 \beta^2 \alpha_2 \beta^2 \alpha_3 \beta^2 \alpha_w \beta^2 \]

Epoch 1

Epoch 2

\[ \beta^2 \alpha_1 \beta^1 \alpha_2 \beta^1 \alpha_3 \beta^1 \alpha_w \beta^1 \]
Malicious Security Compiler for Fluid MPC

- **Max Fluidity**
- **Small State Size**
- **Secure State Transfer**
Conclusion and Open Questions

• Exciting new direction.

• Communication Complexity semi-honest Fluid BGW is $O(n^2 |C|)$.

• Our compiler preserves the fluidity and communication complexity of the underlying semi-honest protocol, but only achieves security with abort.

• Open Questions:
  • Improved efficiency
  • Guaranteed output delivery
  • Exploring other modeling choices
Thank You

aarushig@cs.jhu.edu

https://eprint.iacr.org/2020/754