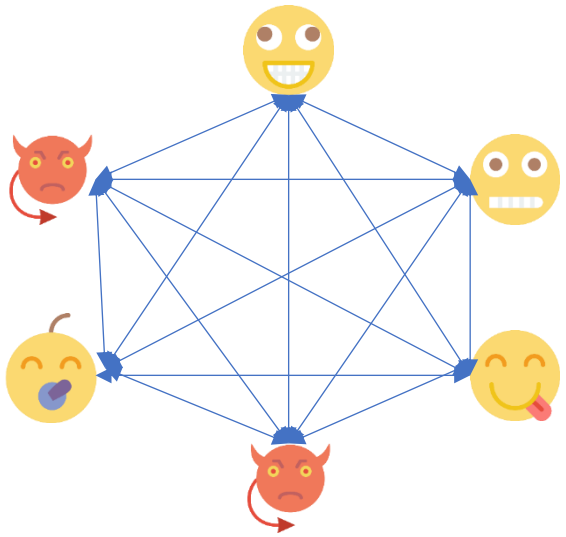
Constant-Round Asynchronous MPC with Optimal Resilience and Linear Communication

Junru Li
Tsinghua University

Yifan Song
Tsinghua University
Shanghai Qi Zhi Institute

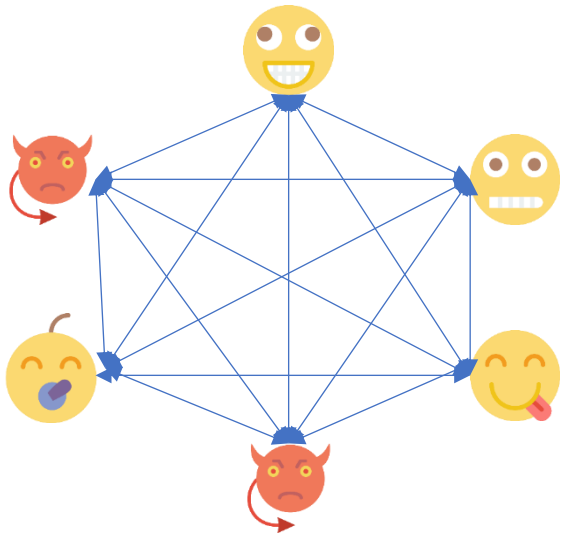
Multiparty Computation



Setting

- Asynchronous Network
- Optimal Resilience $t = (n - 1)/3$
- Fully Malicious Adversary (Security with Abort)

Multiparty Computation



Setting

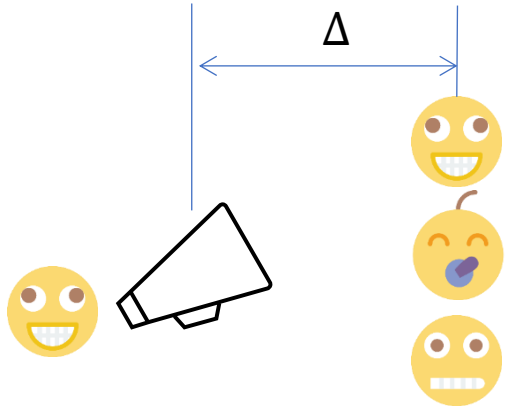
- Asynchronous Network
- Optimal Resilience $t = (n-1)/3$
- Fully Malicious Adversary (Security with Abort)

Target

- Lightweight Cryptographic Primitives (no FHE)
- Constant Round Complexity
- Communication Complexity Linear to n

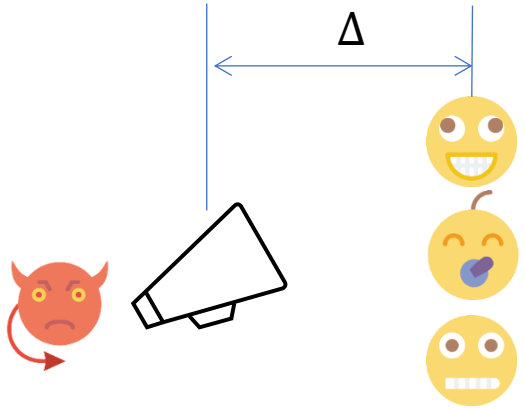
Differences between Sync. and Async.

Sync:



Differences between Sync. and Async.

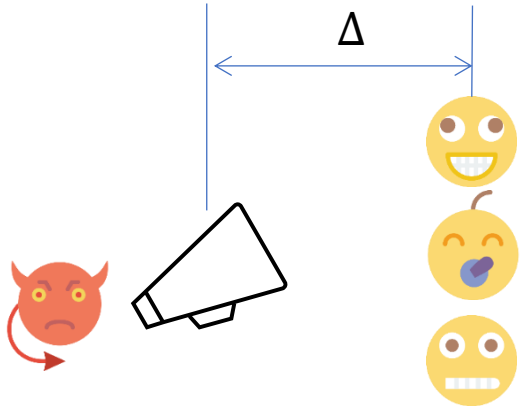
Sync:



If no answer after Δ,
assume ⊥ received

Differences between Sync. and Async.

Sync:

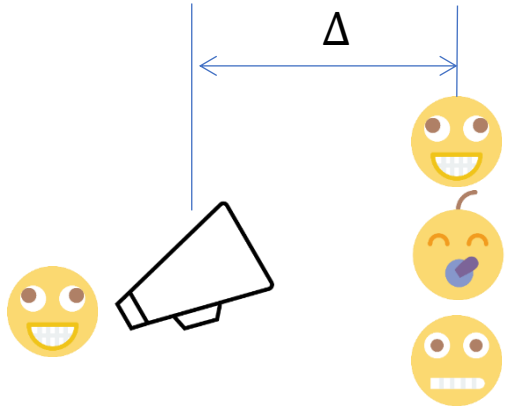


If no answer after Δ,
assume ⊥ received

- Can be realized when $t = n - 1$
- Δ must be large

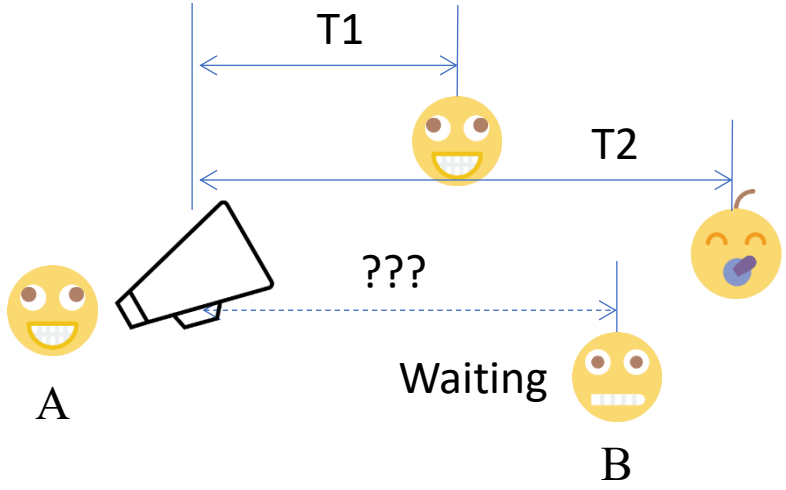
Differences between Sync. and Async.

Sync:



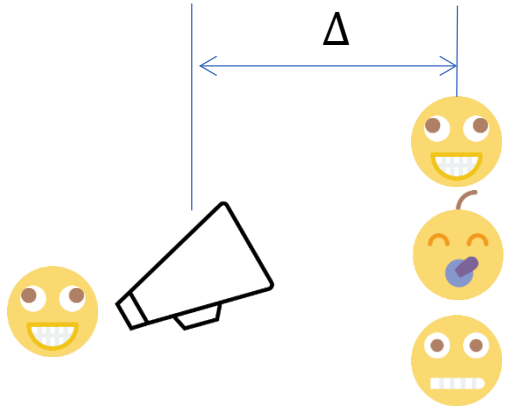
- Can be realized when $t = n - 1$
- Δ must be large

Async:



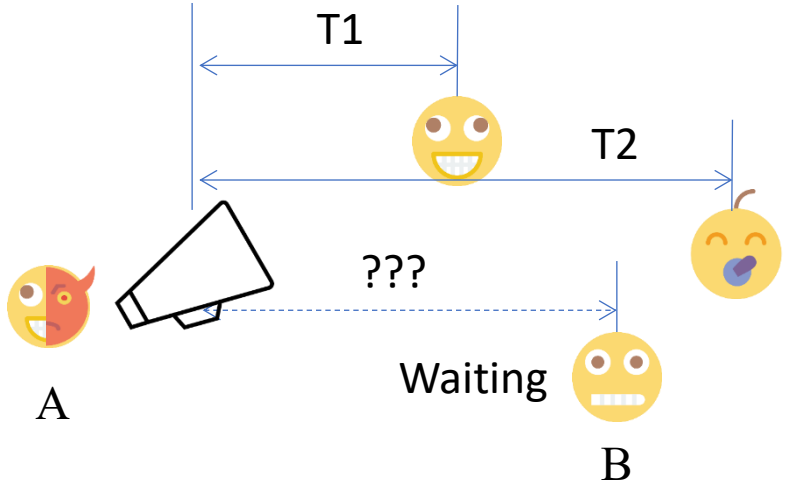
Differences between Sync. and Async.

Sync:



- Can be realized when $t = n - 1$
- Δ must be large

Async:



Cannot distinguish *dishonest sender not sending* vs *slow honest sender*

- Protocol runs at speed of actual network delay
- t parties may not be able to provide inputs
- Requiring $t < n/3$

Known Results from Literatures

Communication-Efficient but Non-Constant-Round AMPC (GOD)

- Perfect: $O(|C|n)$ communication is achieved for when $t < \frac{n}{4}$ [AAPP24].
- Statistical: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ [GLS24], with a large additive overhead $O(n^{14}\kappa^2)$.
- Computational: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ assuming RO [BJK+25], with a low overhead.

Known Results from Literatures

Communication-Efficient but Non-Constant-Round AMPC (GOD)

- Perfect: $O(|C|n)$ communication is achieved for when $t < \frac{n}{4}$ [AAPP24].
- Statistical: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ [GLS24], with a large additive overhead $O(n^{14}\kappa^2)$.
- Computational: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ assuming RO [BJK+25], with a low overhead.

Constant-Round but Communication-Heavy AMPC

- Requiring $\Omega(|C|n^3\kappa)$ communication from OWFs in the $\mathcal{F}_{ACS}$ -hybrid model [CGHZ16].
- Based on BMR template.
- Achieving GOD.

Known Results from Literatures

Communication-Efficient but Non-Constant-Round AMPC (GOD)

- Perfect: $O(|C|n)$ communication is achieved for when $t < \frac{n}{4}$ [AAPP24].
- Statistical: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ [GLS24], with a large additive overhead $O(n^{14}\kappa^2)$.
- Computational: $O(|C|n\kappa)$ communication is achieved for when $t < \frac{n}{3}$ assuming RO [BJK+25], with a low overhead.

Constant-Round but Communication-Heavy AMPC

- Requiring $\Omega(|C|n^3\kappa)$ communication from OWFs in the $\mathcal{F}_{ACS}$ -hybrid model [CGHZ16].
- Based on BMR template.
- Achieving GOD.

The parties must agree on a common set that provide inputs to the MPC, and this process cannot be constant-round in the plain model

Our Result

Assuming random oracles, there exists a computationally secure (with abort) constant-round AMPC in the $\mathcal{F}_{ACS}$ -hybrid model against a fully malicious adversary controlling up to $t < n/3$ parties with communication $O(|C|n\kappa + D(n+\kappa)^2 n\kappa + \mathrm{poly}(n,\kappa))$ plus 3 invocations of $\mathcal{F}_{ACS}$, where $|C|$ is the circuit size, D is the circuit depth, and κ is the computational security parameter.

Our Result

Assuming random oracles, there exists a computationally secure (with abort) constant-round AMPC in the $\mathcal{F}_{ACS}$ -hybrid model against a fully malicious adversary controlling up to $t < n/3$ parties with communication $O(|C|n\kappa + D(n+\kappa)^2 n\kappa + \mathrm{poly}(n,\kappa))$ plus 3 invocations of $\mathcal{F}_{ACS}$, where $|C|$ is the circuit size, D is the circuit depth, and κ is the computational security parameter.

Basic Idea:

- Multiparty garbling
- Send the garbled circuit to all the parties

Our Result

Assuming random oracles, there exists a computationally secure (with abort) constant-round AMPC in the $\mathcal{F}_{ACS}$ -hybrid model against a fully malicious adversary controlling up to $t < n/3$ parties with communication $O(|C|n\kappa + D(n+\kappa)^2 n\kappa + \text{poly}(n,\kappa))$ plus 3 invocations of $\mathcal{F}_{ACS}$, where $|C|$ is the circuit size, D is the circuit depth, and κ is the computational security parameter.

Basic Idea:

- Multiparty garbling
- Send the garbled circuit to all the parties

A single evaluator may never send the outputs, and the parties cannot decide whether the evaluator is corrupted or the network delay is large

Our Result

Assuming random oracles, there exists a computationally secure (with abort) constant-round AMPC in the $\mathcal{F}_{ACS}$ -hybrid model against a fully malicious adversary controlling up to $t < n/3$ parties with communication $O(|C|n\kappa + D(n+\kappa)^2 n\kappa + \text{poly}(n,\kappa))$ plus 3 invocations of $\mathcal{F}_{ACS}$, where $|C|$ is the circuit size, D is the circuit depth, and κ is the computational security parameter.

Basic Idea:

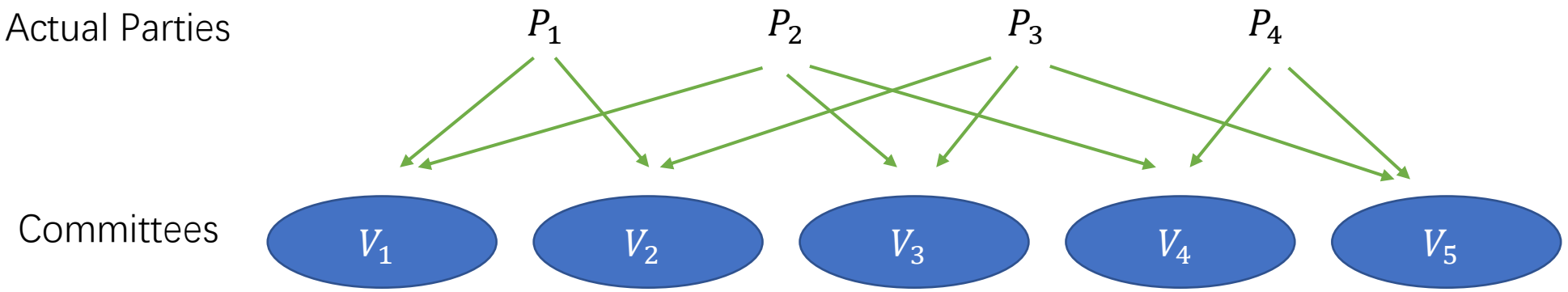
- Multiparty garbling
- Send the garbled circuit to all the parties

A single evaluator may never send the outputs, and the parties cannot decide whether the evaluator is corrupted or the network delay is large

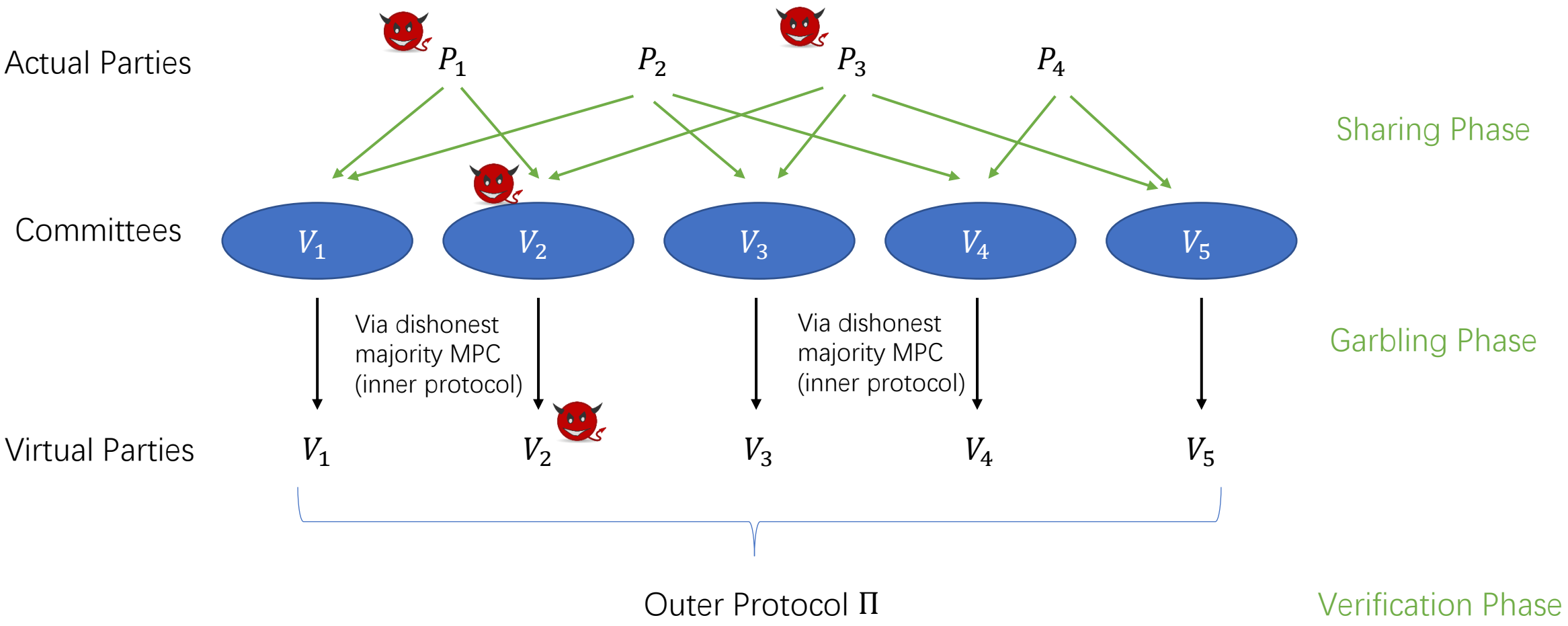
Requiring an $O(|C|\kappa)$-size multiparty garbled circuit (omitting the $D \cdot \text{poly}(n,\kappa)$ term):

- The only known construction in the synchronous case is [GLOS25]

Multiparty Garbling of [GLOS25]



Multiparty Garbling of [GLOS25]



Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

Difficulties

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

2. Garbling Phase: Run the inner protocols to garble the parties' local computations for the underlying protocol

Difficulties

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

2. Garbling Phase: Run the inner protocols to garble the parties' local computations for the underlying protocol

Difficulties

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

Dishonest majority asynchronous protocol does not exist.

Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

2. Garbling Phase: Run the inner protocols to garble the parties' local computations for the underlying protocol

3. Verification Phase: Open a small fraction of virtual parties' views

Difficulties

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

Dishonest majority asynchronous protocol does not exist.

Run the Protocol in Asynchronous Network

Protocol Steps

1. Sharing Phase: All parties distribute preprocessing/input sharings to virtual parties

2. Garbling Phase: Run the inner protocols to garble the parties' local computations for the underlying protocol

3. Verification Phase: Open a small fraction of virtual parties' views

Difficulties

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

Dishonest majority asynchronous protocol does not exist.

A corrupted party may never open his commitment and view when he is checked.

Difficulties Caused by the Asynchrony

1. Generating preprocessing/input sharings: A corrupted party may send his input sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.
   - Previous solution: ACSS (but only for Shamir sharings)

2. MPC-in-the-head Verification: A corrupted party may never open his commitments and view when he is checked.

3. Inner protocols: Dishonest majority asynchronous protocol does not exist.

Difficulties Caused by the Asynchrony

1. Generating preprocessing/input sharings: A corrupted party may send his input sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.
   - Previous solution: ACSS (but only for Shamir sharings)

2. MPC-in-the-head Verification: A corrupted party may never open his commitments and view when he is checked.

Solution: Use Asynchronous Verifiable Information Dispersal (AVID) [CT05, ADD+22]

Can be instantiated from RO

3. Inner protocols: Dishonest majority asynchronous protocol does not exist.

Solution: Run a synchronous inner protocol

Asynchronous Verifiable Information Dispersal

Goal: A $\xrightarrow{M}$ B

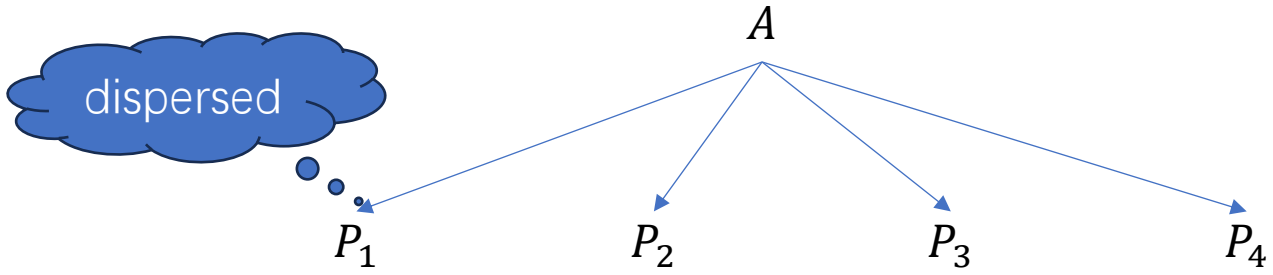
All the parties know M is sent

Asynchronous Verifiable Information Dispersal

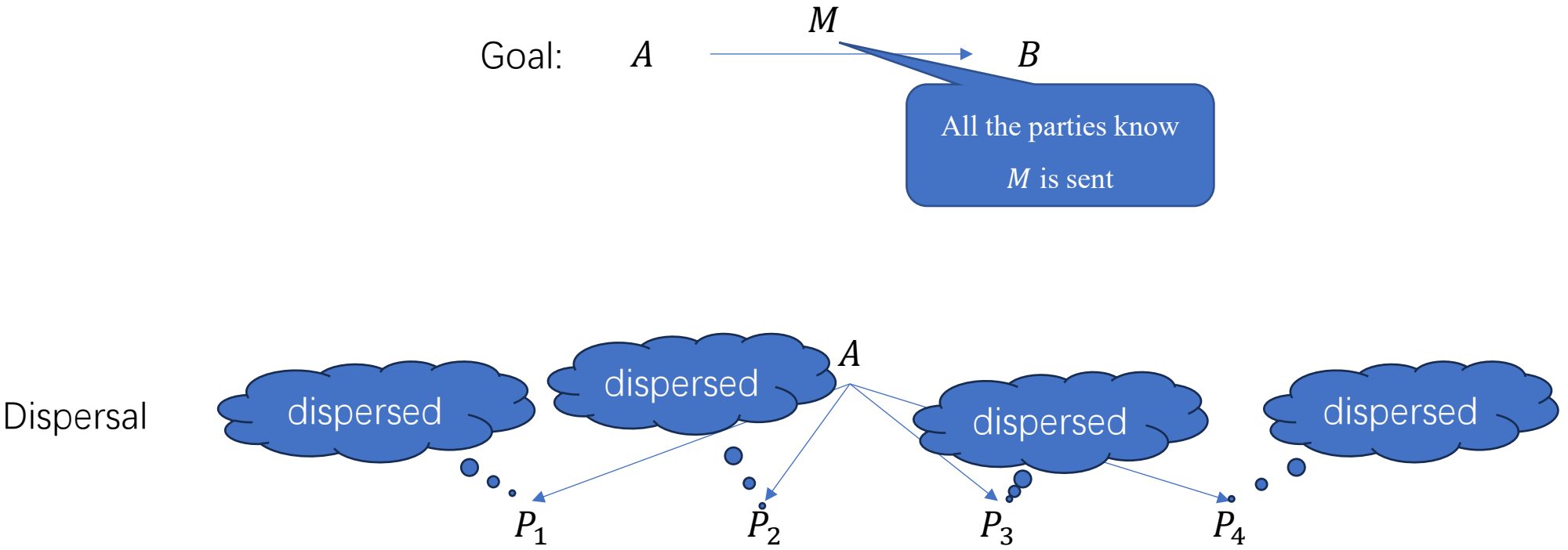
Goal: A M B

All the parties know M is sent

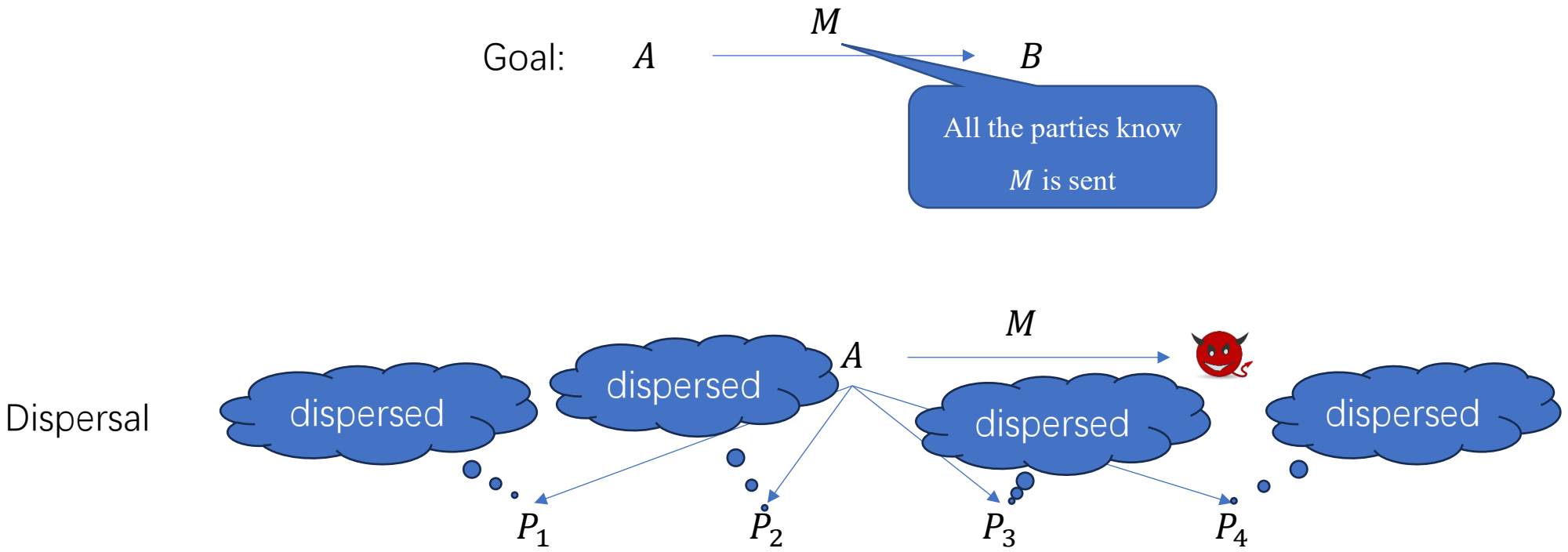
Dispersal



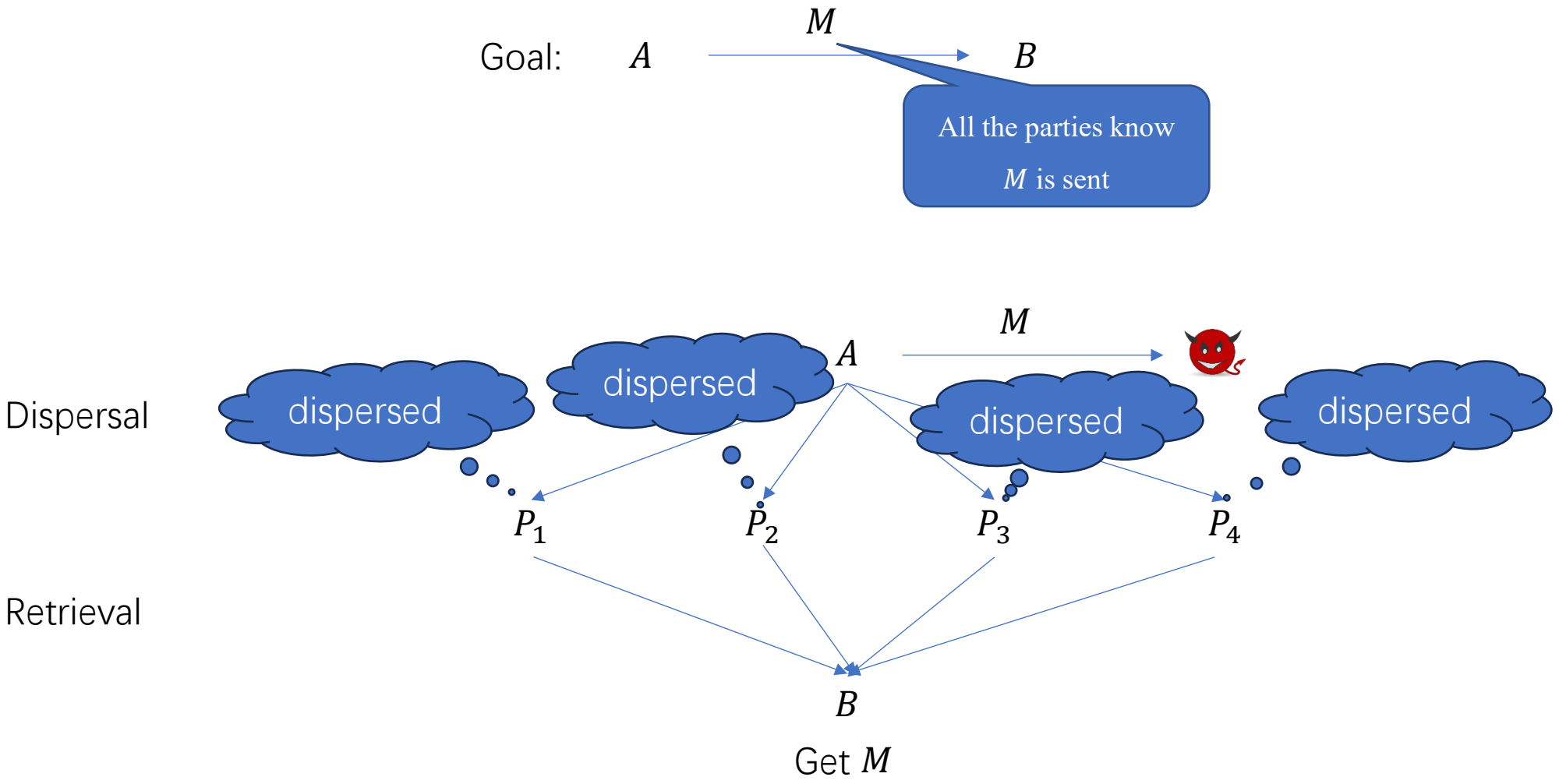
Asynchronous Verifiable Information Dispersal



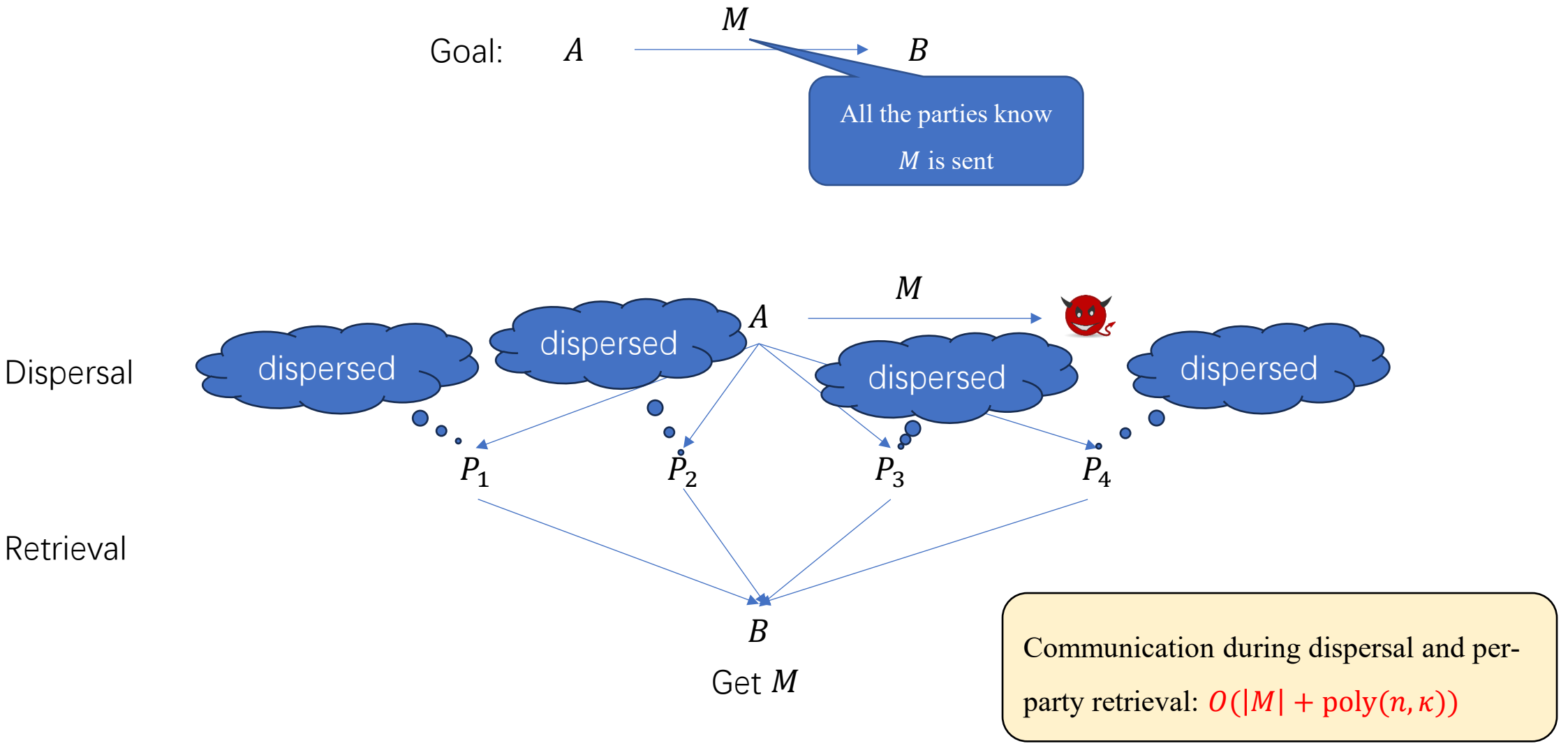
Asynchronous Verifiable Information Dispersal



Asynchronous Verifiable Information Dispersal

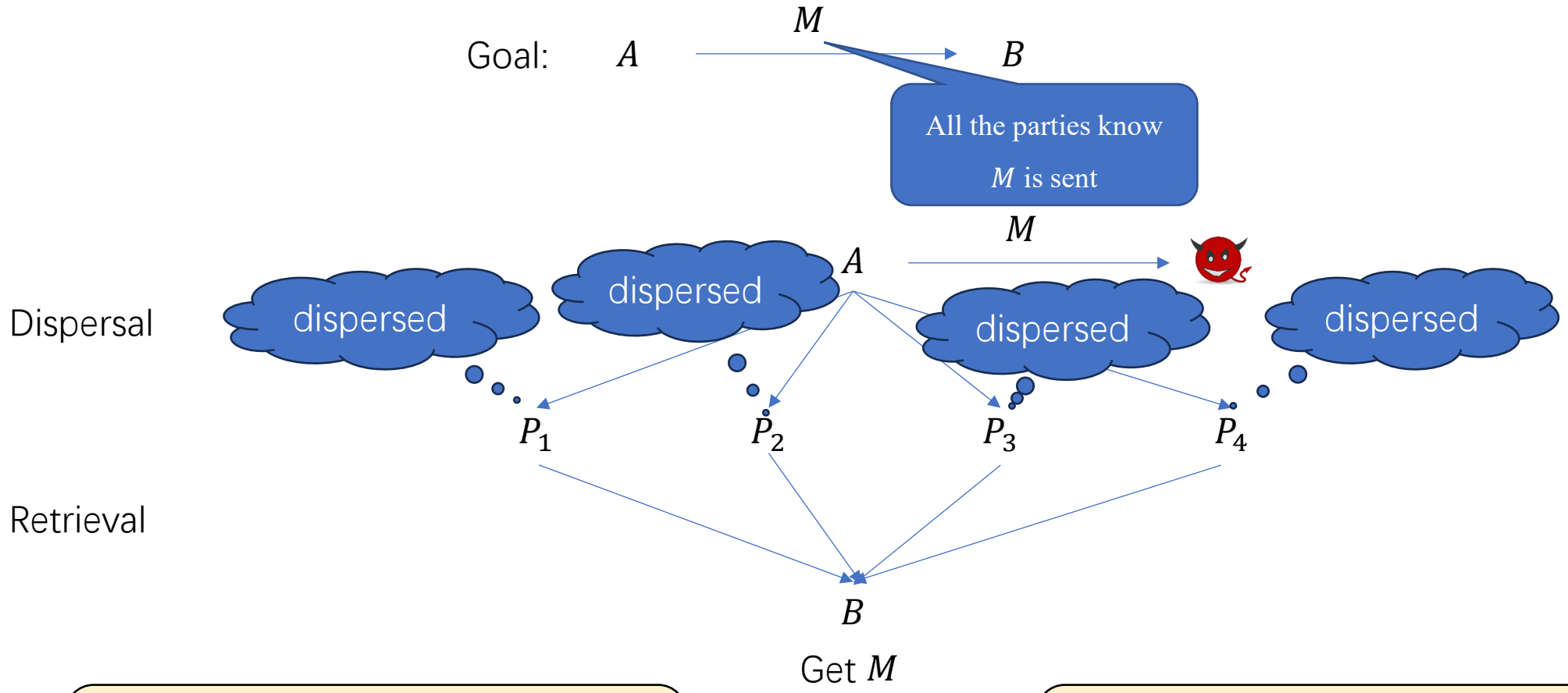


Asynchronous Verifiable Information Dispersal



Communication during dispersal and per-party retrieval: $O(|M| + \text{poly}(n, \kappa))$

Asynchronous Verifiable Information Dispersal



Communication during dispersal and per-party retrieval: $O(|M| + \text{poly}(n, \kappa))$

Only $O(\text{poly}(n, \kappa))$ overhead for sending all messages of a constant-round MPC via AVID

Generating Sharings via AVID

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

1. Preparing pair-wise symmetric keys.

2. Send the ciphertexts for the shares via AVID.

Generating Sharings via AVID

A corrupted party may send his sharings to only a part of the parties, and the rest of the parties may wait forever for the shares.

1. Preparing pair-wise symmetric keys.
   - Via a general constant-round AMPC, only requiring $O(\text{poly}(n, \kappa))$ communication

2. Send the ciphertexts for the shares via AVID.

Generating Commitments via AVID

Commit:

1. Preparing a secret-shared seed (using a general constant-round ACSS).
2. Reconstruct the seed to the sender.
3. Mask the message using the seed (via RO).
4. Disperse the masked messages via AVID.

Generating Commitments via AVID

Commit:

1. Preparing a secret-shared seed (using a general constant-round ACSS).
2. Reconstruct the seed to the sender.
3. Mask the message using the seed (via RO).
4. Disperse the masked messages via AVID.

Open to a Party:

1. Reconstructing the seed to the party.
2. Let the party retrieve the masked message.
3. Decrypt the message using the seed.

Inner Protocols

Observation 1: We don't need all the virtual parties' garbled circuits
(only need enough garbled circuits for reconstructions of the label shares)

Inner Protocols

Observation 1: We don't need all the virtual parties' garbled circuits
(only need enough garbled circuits for reconstructions of the label shares)

- **Not all honest virtual parties are required to terminate the inner protocol**

Inner Protocols

Observation 1: We don't need all the virtual parties' garbled circuits
(only need enough garbled circuits for reconstructions of the label shares)

- **Not all honest virtual parties are required to terminate the inner protocol**

Observation 2: Without guaranteed termination, a synchronous protocol can run in the asynchronous setting

Inner Protocols

Observation 1: We don't need all the virtual parties' garbled circuits
(only need enough garbled circuits for reconstructions of the label shares)

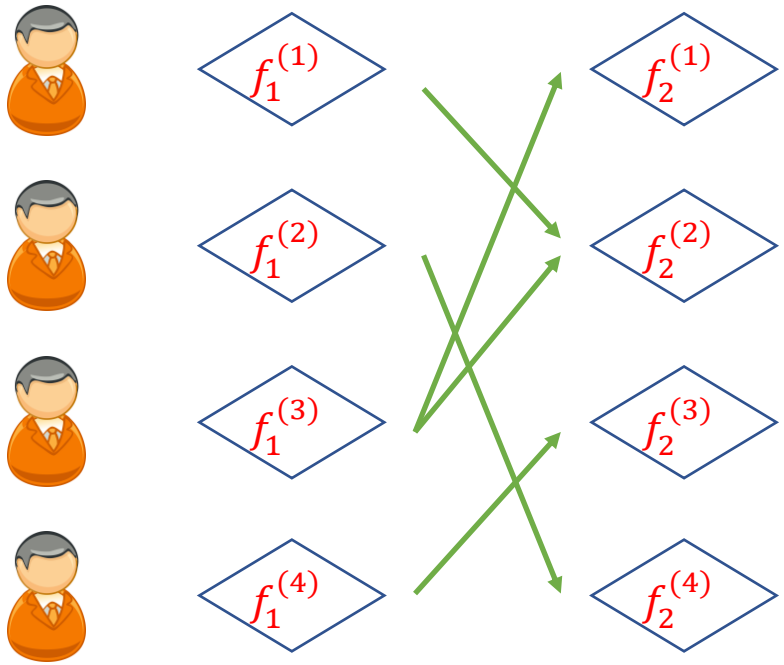
- **Not all honest virtual parties are required to terminate the inner protocol**

Observation 2: Without guaranteed termination, a synchronous protocol can run in the asynchronous setting

Idea: We can run synchronous inner protocols

Inner Protocols

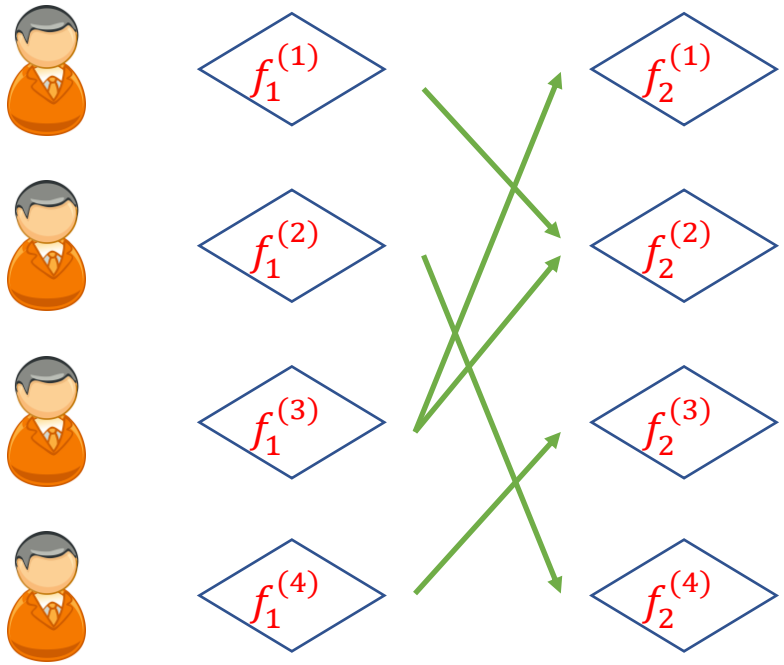
Run a synchronous protocol in the asynchronous setting



A Synchronous Round

Inner Protocols

Run a synchronous protocol in the asynchronous setting



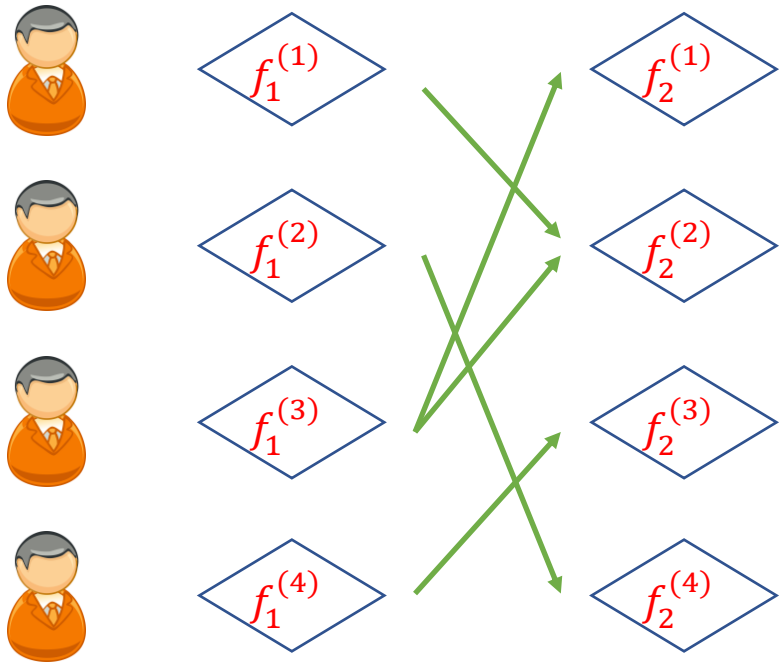
A Synchronous Round

Properties of a synchronous round:

- Can receive all the messages in a round
- When a round finishes, everyone knows.

Inner Protocols

Run a synchronous protocol in the asynchronous setting



A Synchronous Round

Properties of a synchronous round:

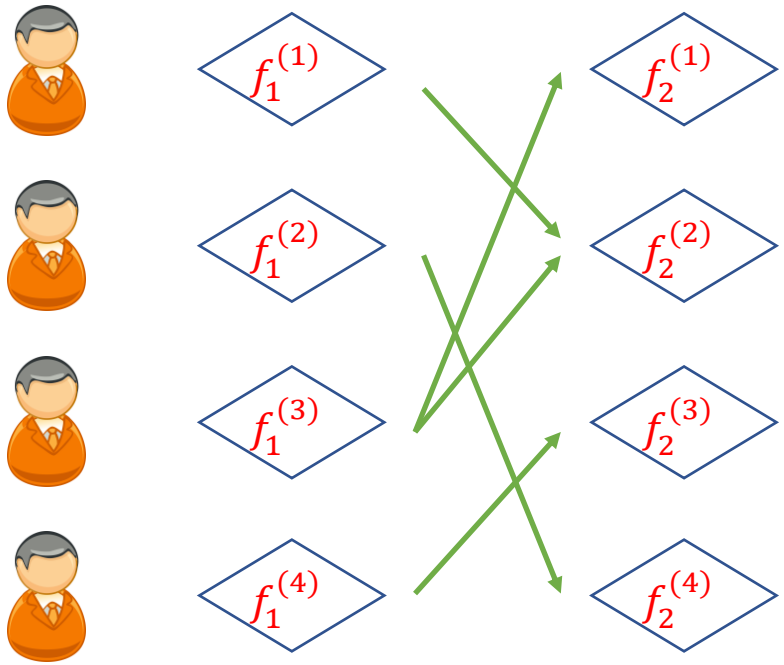
- Can receive all the messages in a round
- When a round finishes, everyone knows.

Solution:

- Use AVID + encryption to send messages
- Wait for all the dispersal signals in one round and then continue

Inner Protocols

Run a synchronous protocol in the asynchronous setting



A Synchronous Round

Properties of a synchronous round:

- Can receive all the messages in a round
- When a round finishes, everyone knows.

Solution:

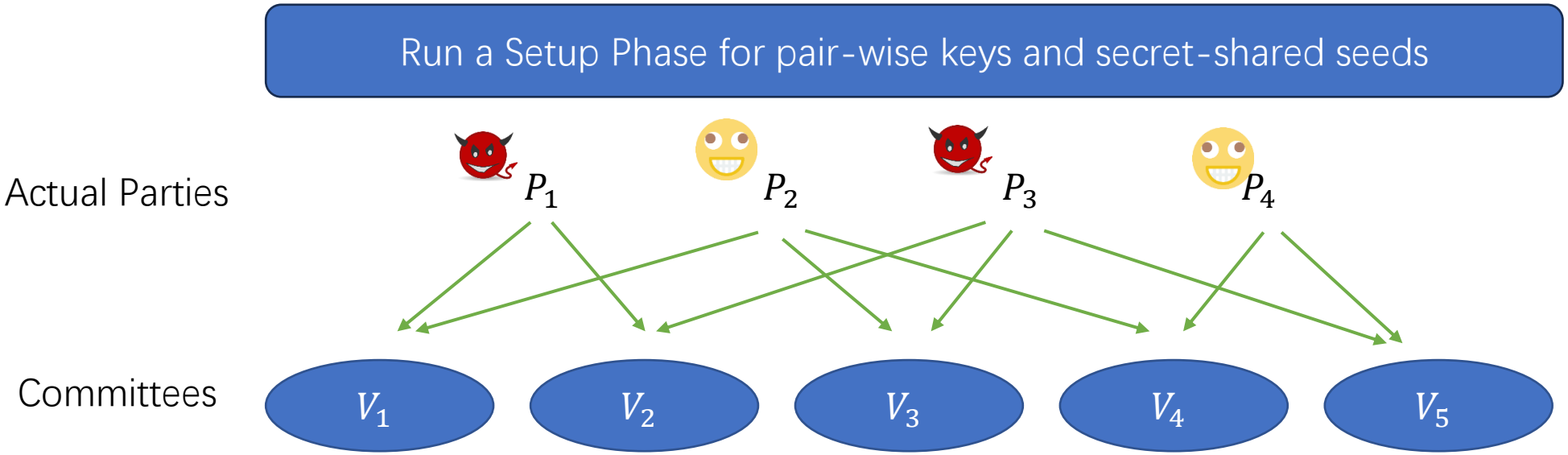
- Use AVID + encryption to send messages
- Wait for all the dispersal signals in one round and then continue

Also providing commitments to the view of virtual parties

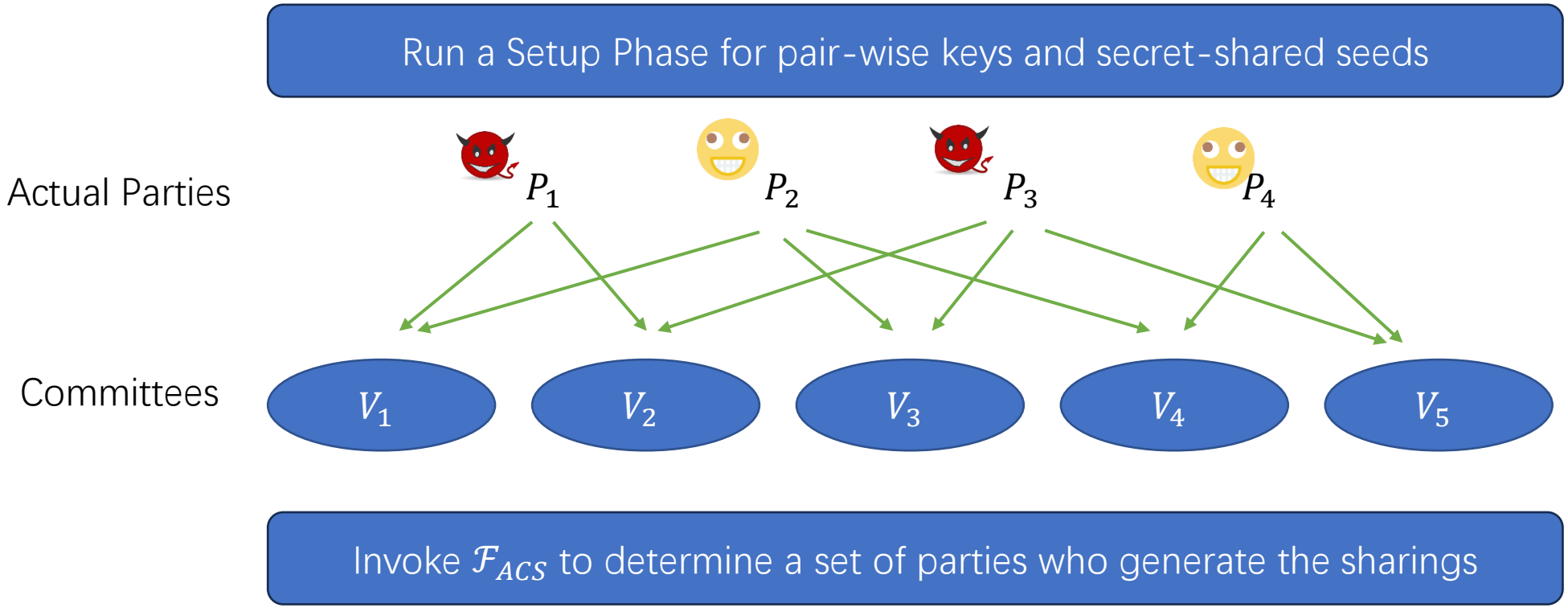
Multiparty Garbling Outline

Run a Setup Phase for pair-wise keys and secret-shared seeds

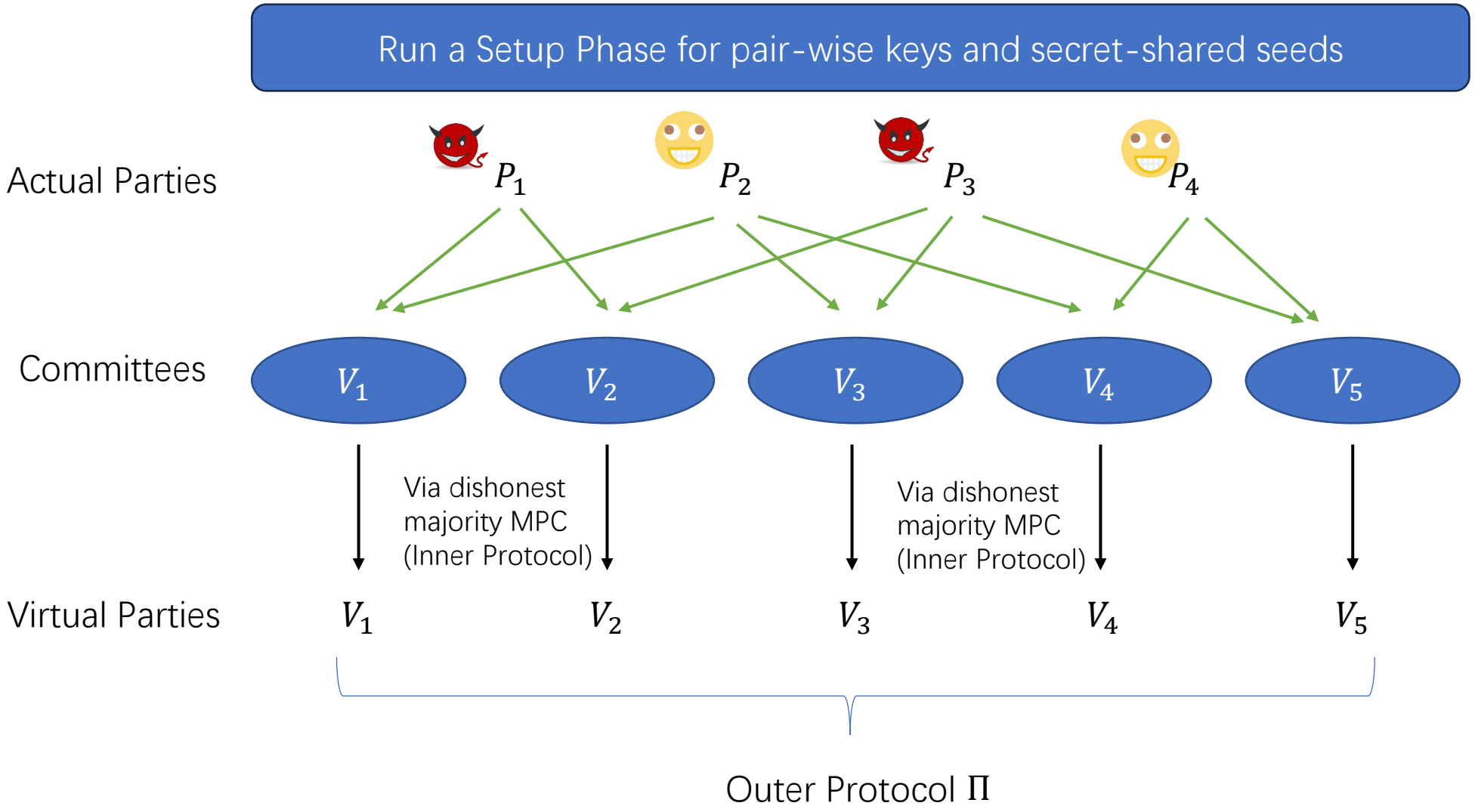
Multiparty Garbling Outline



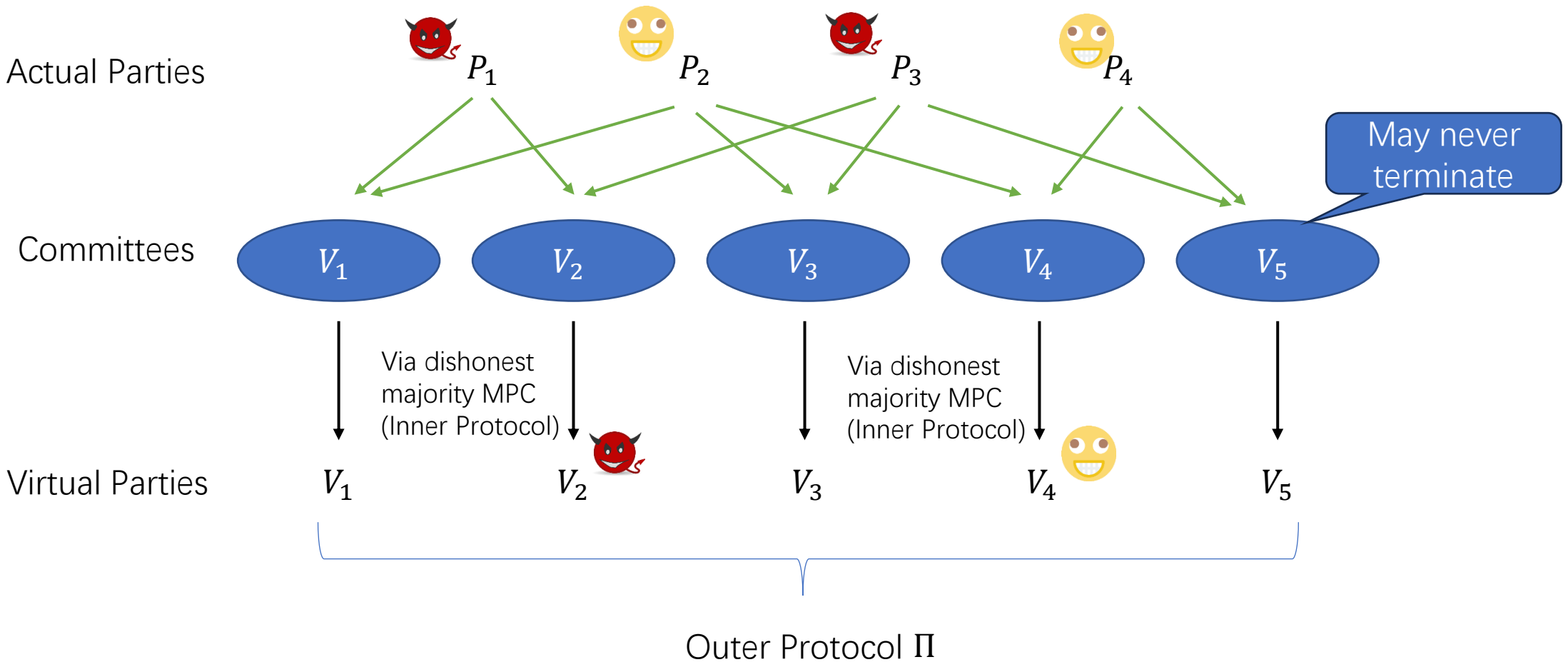
Multiparty Garbling Outline



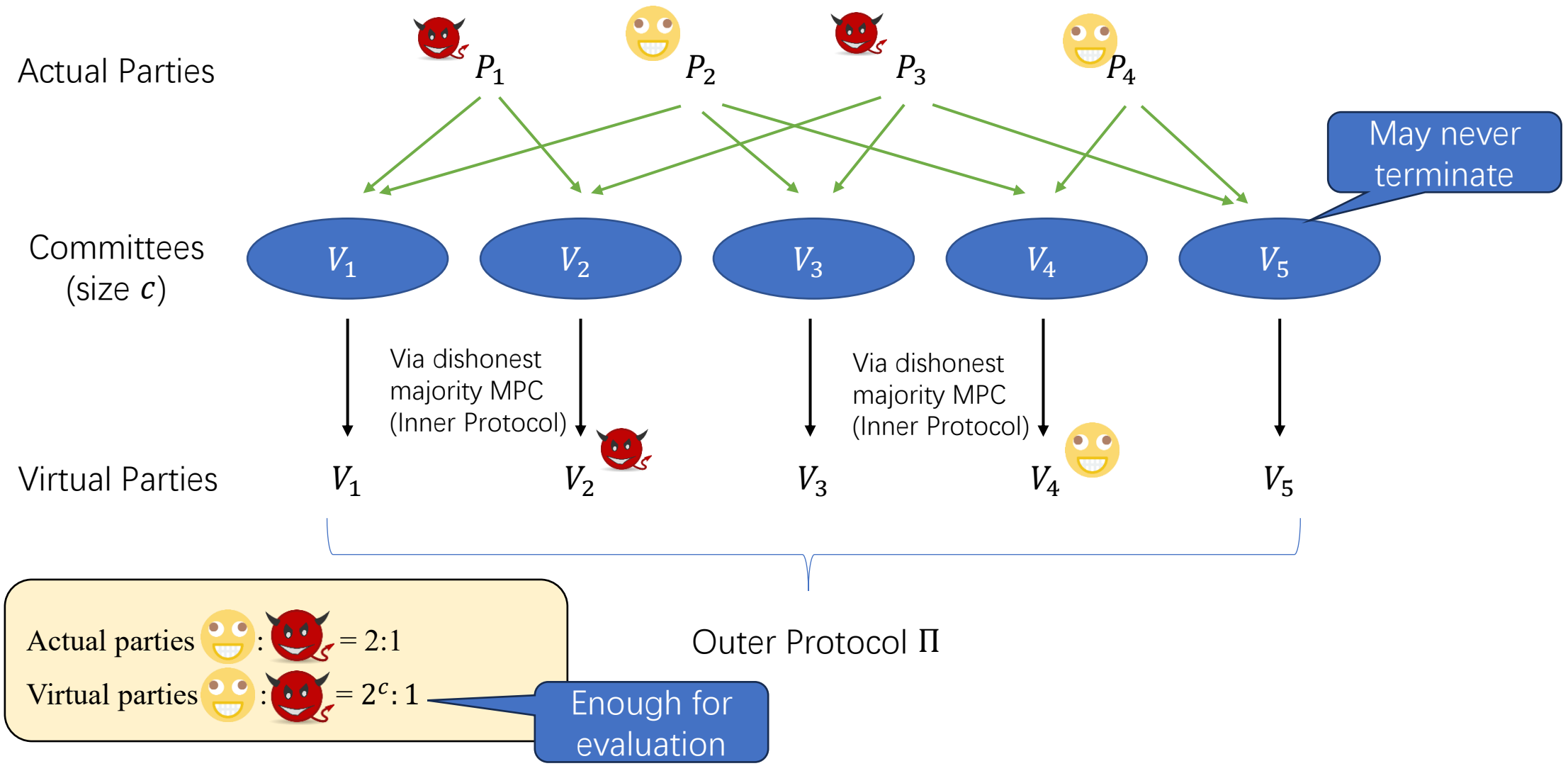
Multiparty Garbling Outline



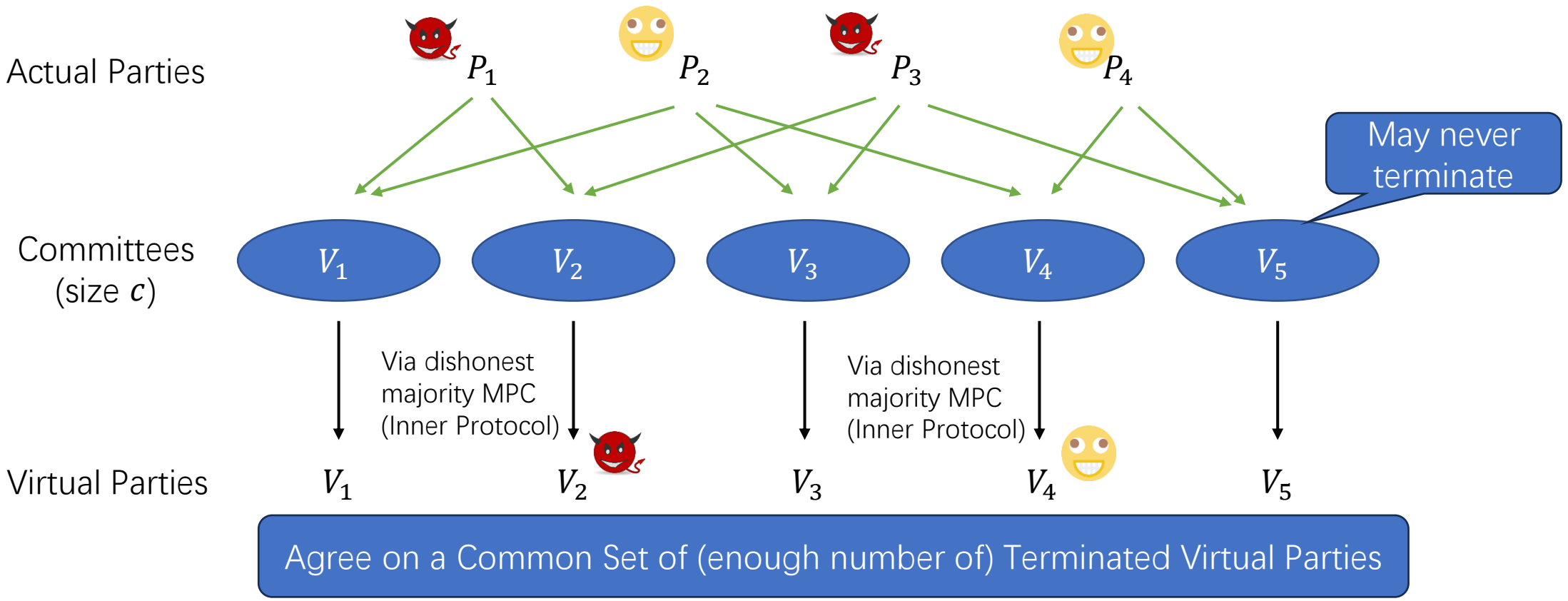
Multiparty Garbling Outline



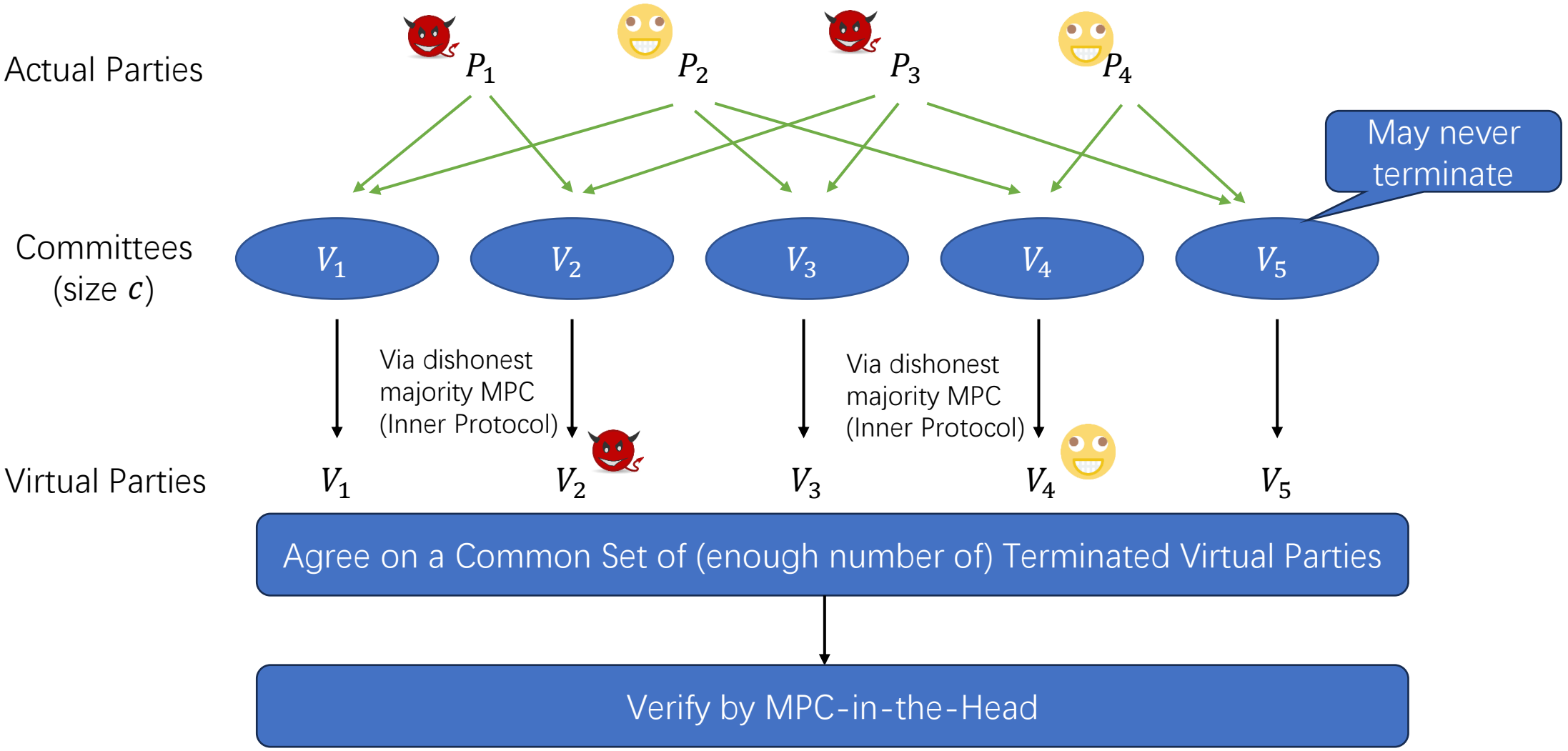
Multiparty Garbling Outline



Multiparty Garbling Outline



Multiparty Garbling Outline



Thanks!

https://eprint.iacr.org/2025/1032