Asynchronous Multi-Party Computation

Vassilis Zikas

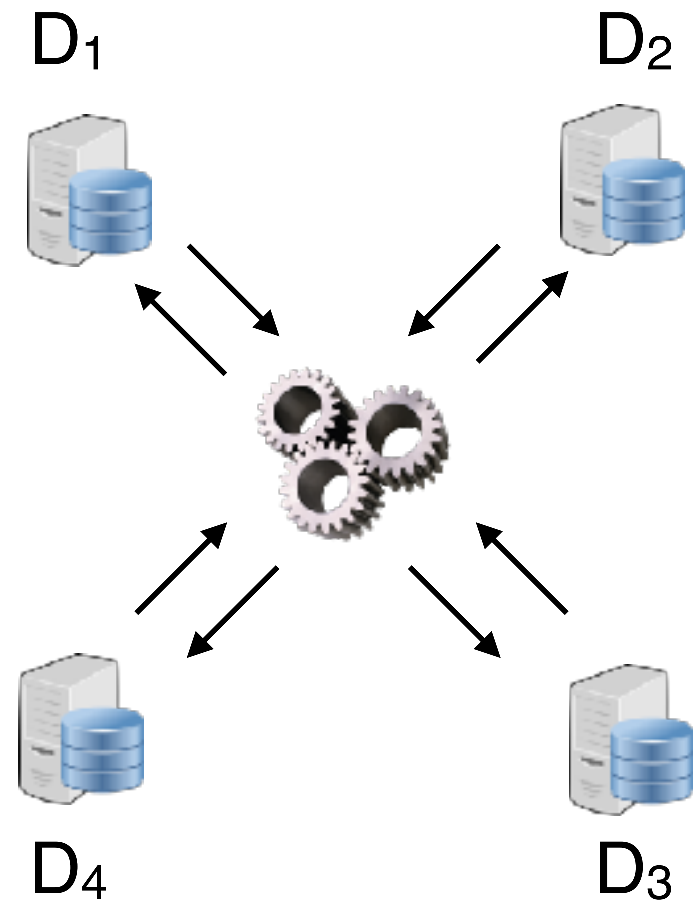
RPI

MPC School

IIT Mumbai

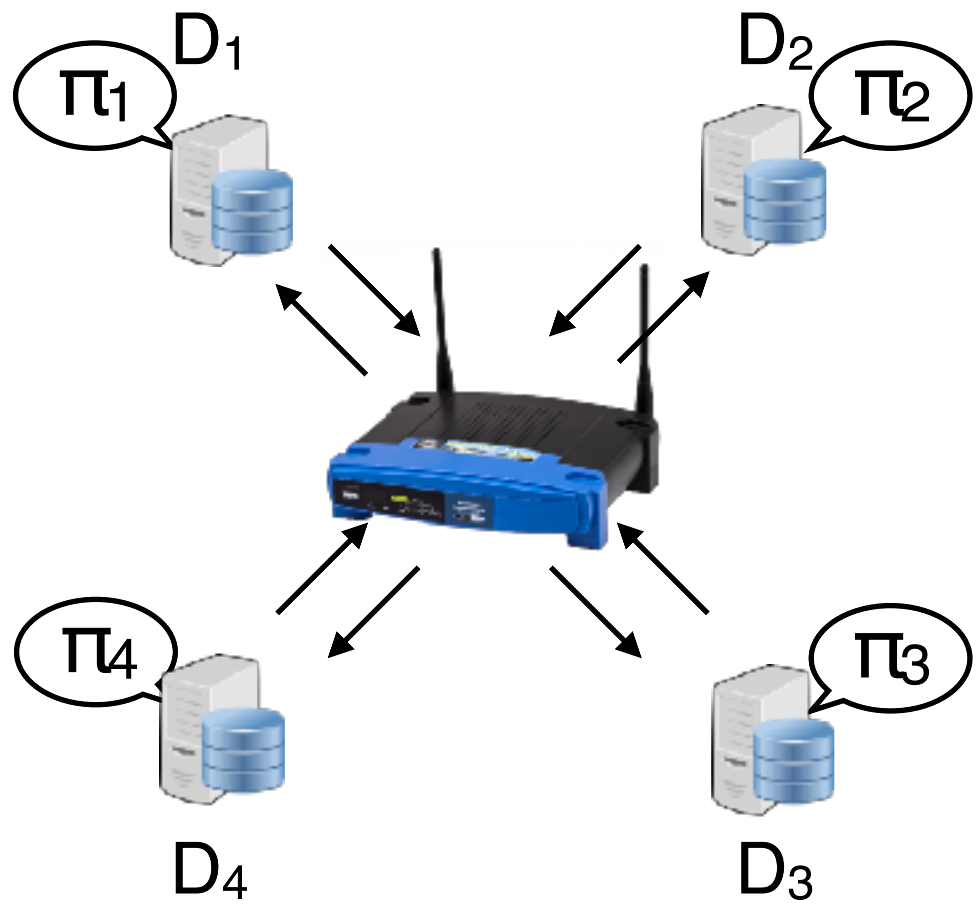
Secure Multi-Party Computation (MPC)

Security



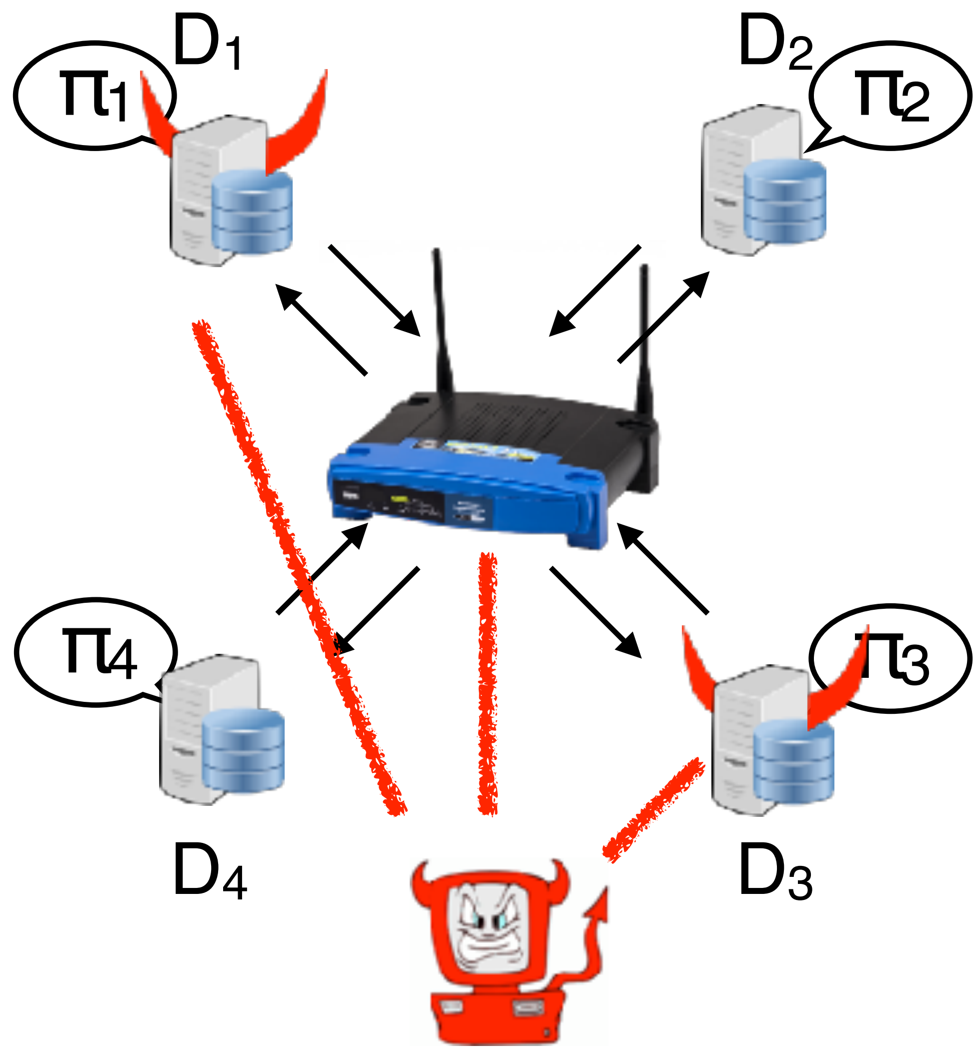
Secure Multi-Party Computation (MPC)

Security



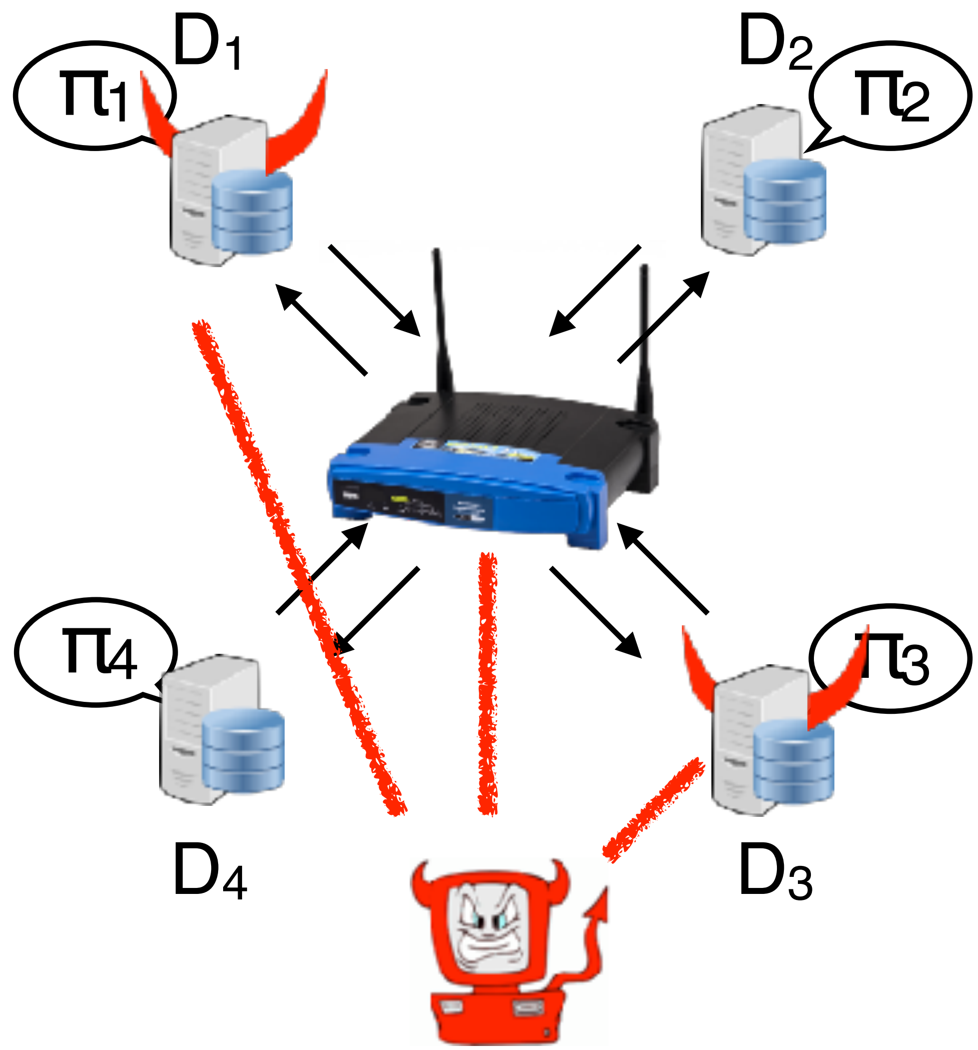
Secure Multi-Party Computation (MPC)

Security



Secure Multi-Party Computation (MPC)

Security

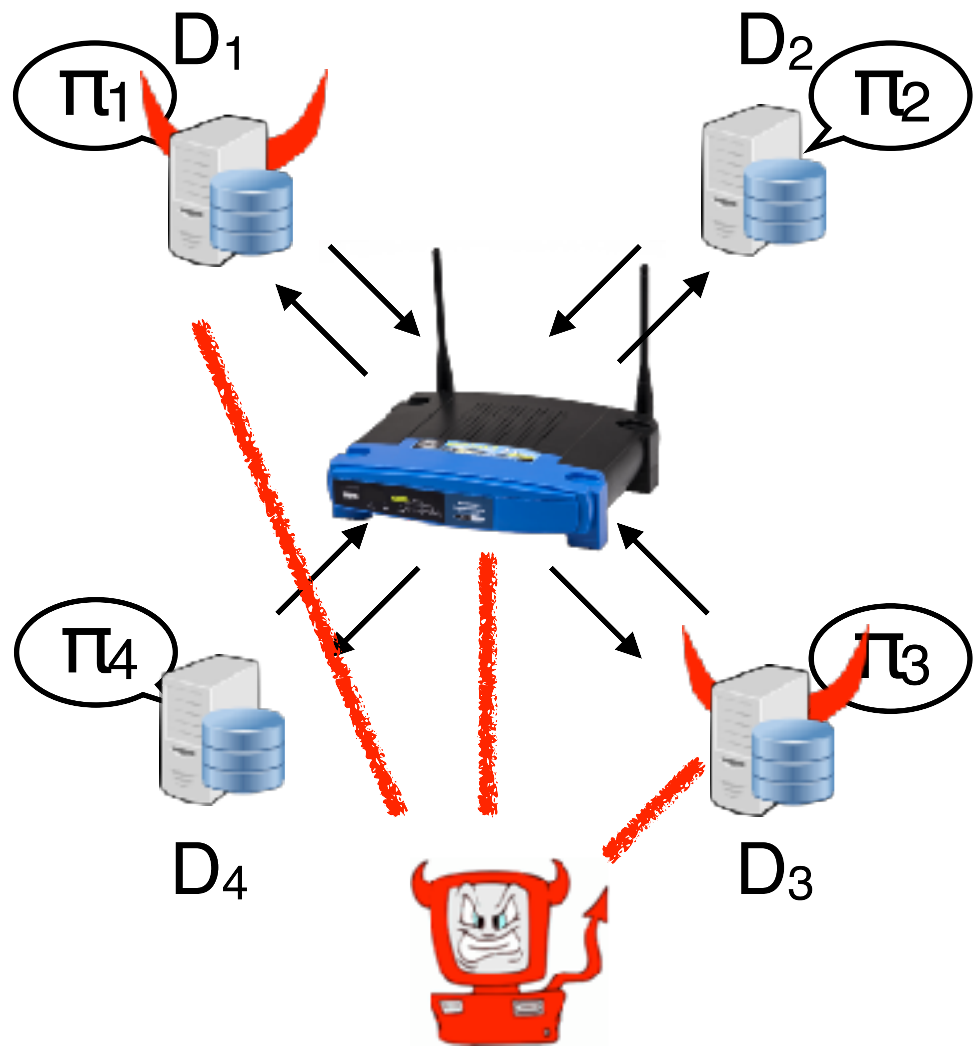


Protocol π is secure if for every adversary:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties learn their correct outputs

Secure Multi-Party Computation (MPC)

Security

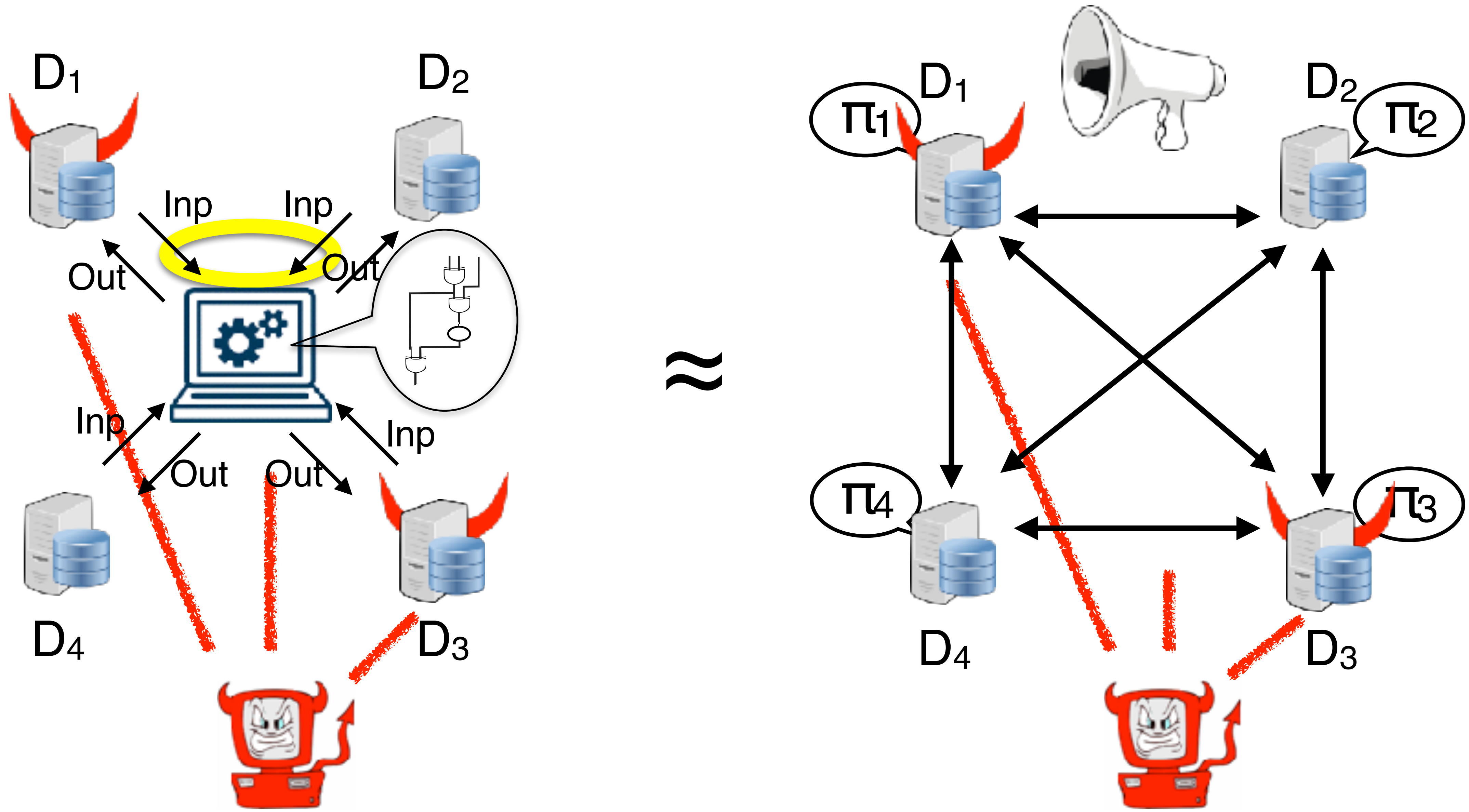


Protocol π is secure if for every adversary:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties learn their correct outputs
- *(termination)* The protocol terminates after a finite number of rounds

Secure Multi-Party Computation (MPC)

Ideal World: Specification   Real World: Protocol



Model

- n players
- Computation over $(\mathbb{F}, \oplus, \otimes)$ — E.g. $(\mathbb{Z}_p, +, \cdot)$
- Communication: Point-to-point secure channels (and Broadcast)
- $\boxed{\text{Synchrony: Messages sent in round i are delivered by round i+1}}$

The Synchronous model

Multi-Party Computation [GMW87, BGW88, CCD88, RB89, CDDHR99, ...]
Byzantine Agreement [PSL80,BGP89,DS82, FL82, TPS87, FM88, BPW91, ...]
...

Round Structure

- Round *r:* parties read round *r-1* messages and compute/send round r messages.
- Round *r-1* messages are guaranteed to be delivered by beginning of Round *r*

The Synchronous model

Multi-Party Computation [GMW87, BGW88, CCD88, RB89, CDDHR99, ...]
Byzantine Agreement [PSL80,BGP89,DS82, FL82, TPS87, FM88, BPW91, ...]
...

Round Structure

- Round *r:* parties read round *r-1* messages and compute/send round r messages.
- Round *r-1* messages are guaranteed to be delivered by beginning of Round *r*

Real-world Assumptions:

- Channels with known bounded delay
- (Partially) Synchronized clocks

The Synchronous model

Multi-Party Computation [GMW87, BGW88, CCD88, RB89, CDDHR99, ...]
Byzantine Agreement [PSL80,BGP89,DS82, FL82, TPS87, FM88, BPW91, ...]
...

Round Structure

- Round *r:* parties read round *r-1* messages and compute/send round r messages.
- Round *r-1* messages are guaranteed to be delivered by beginning of Round *r*

Real-world Assumptions:

- Channels with known bounded delay
- (Partially) Synchronized clocks

Idea:
Use clocks to wait sufficiently long (at least network latency)

The Synchronous model

Multi-Party Computation [GMW87, BGW88, CCD88, RB89, CDDHR99, ...]
Byzantine Agreement [PSL80,BGP89,DS82, FL82, TPS87, FM88, BPW91, ...]
...

Round Structure

- Round *r:* parties read round *r-1* messages and compute/send round r messages.
- Round *r-1* messages are guaranteed to be delivered by beginning of Round *r*

Real-world Assumptions:

- Channels with known bounded delay
- (Partially) Synchronized clocks

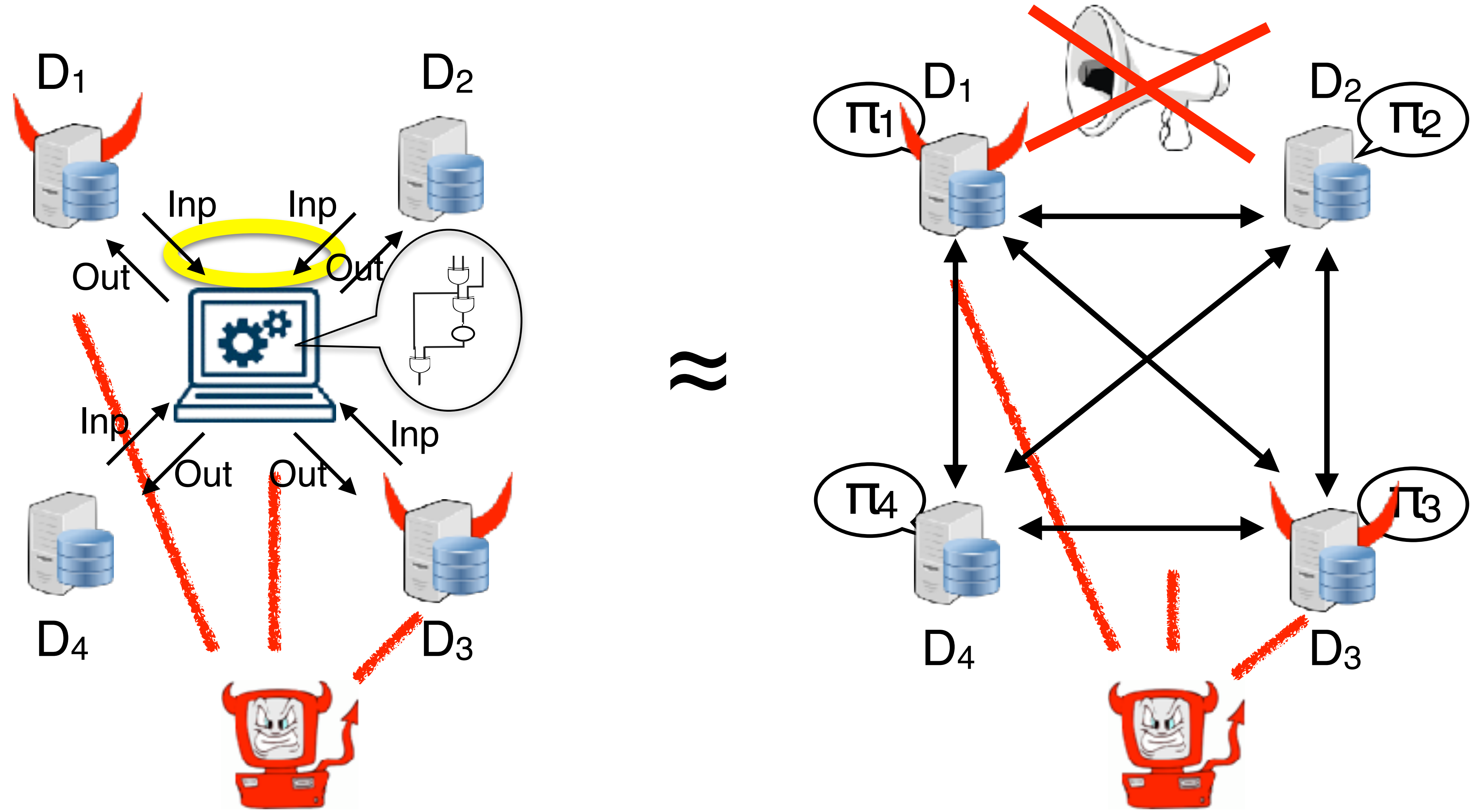
Idea:
Use clocks to wait sufficiently long (at least network latency)

Security Guarantees (in reality)

- Correctness, Privacy, ...
- **Input Completeness:** the inputs of all honest parties are considered
- (**Guaranteed) termination:** In the time corresponding to the end of the last round, the protocol terminates (independent of adversary).

The Asynchronous Model

Ideal World: Specification　　　　Real World: Protocol

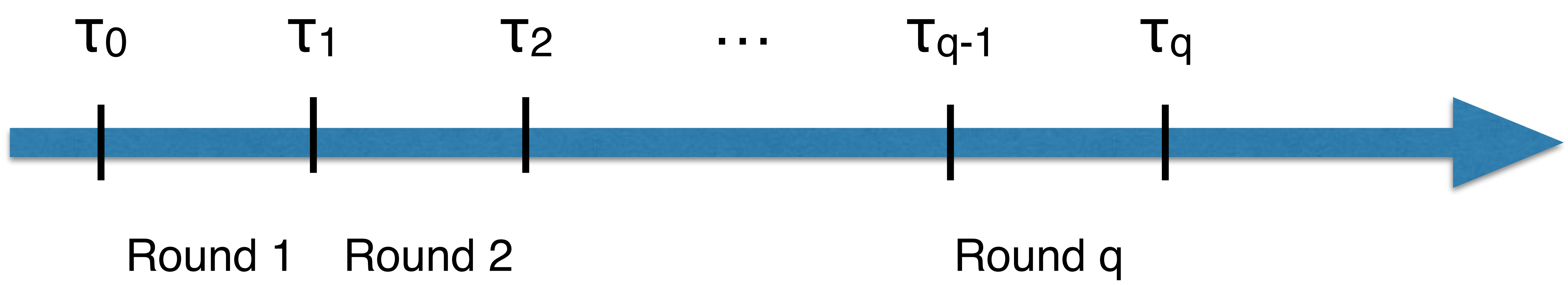


Model

- n players
- Computation over ($\mathbb{F}$, ⊕, ⊗) — E.g. ($\mathbb{Z}_p$, + , ·)
- Communication: Point-to-point secure channels ~~(and Broadcast)~~
- ~~Synchrony: Messages sent in round i are delivered by round i+1~~

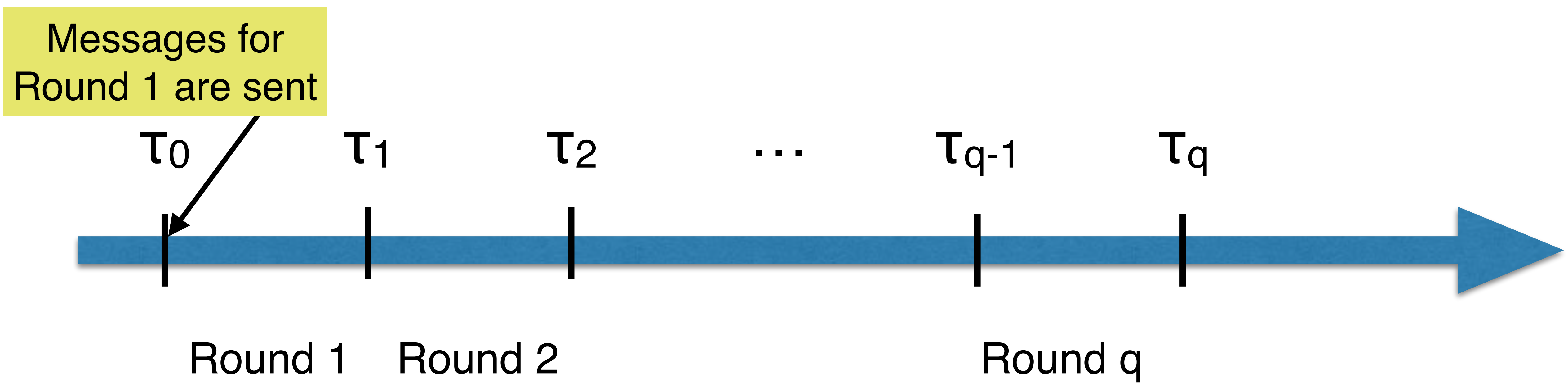
Why Asynchronous Computation?

Timeline of a Synchronous protocol



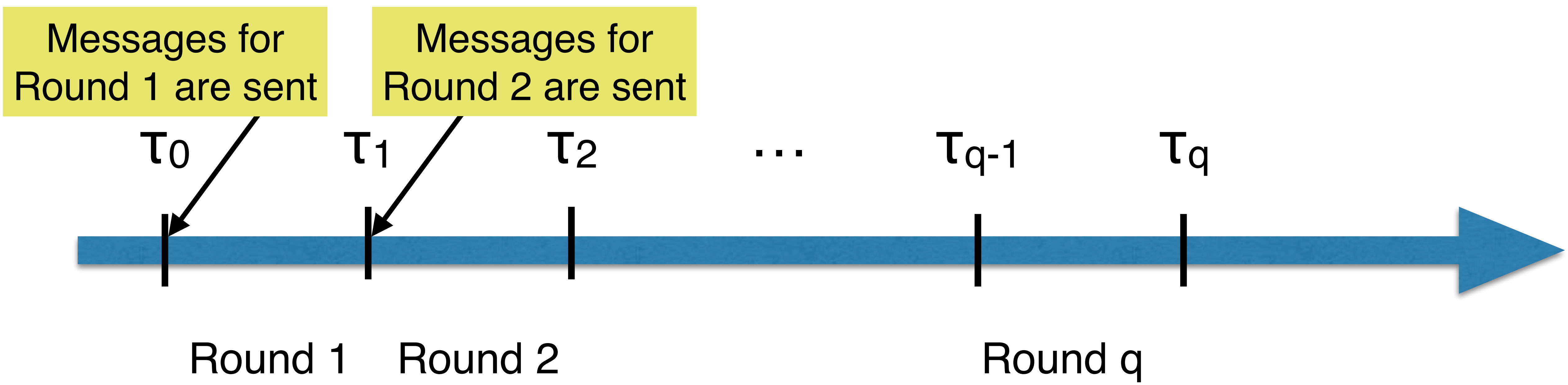
Why Asynchronous Computation?

Timeline of a Synchronous protocol



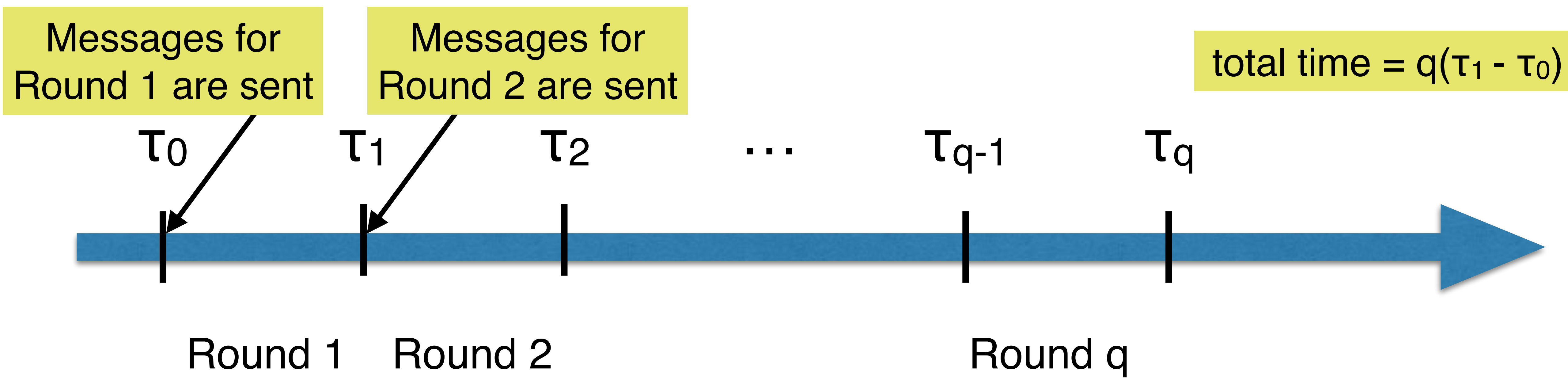
Why Asynchronous Computation?

Timeline of a Synchronous protocol



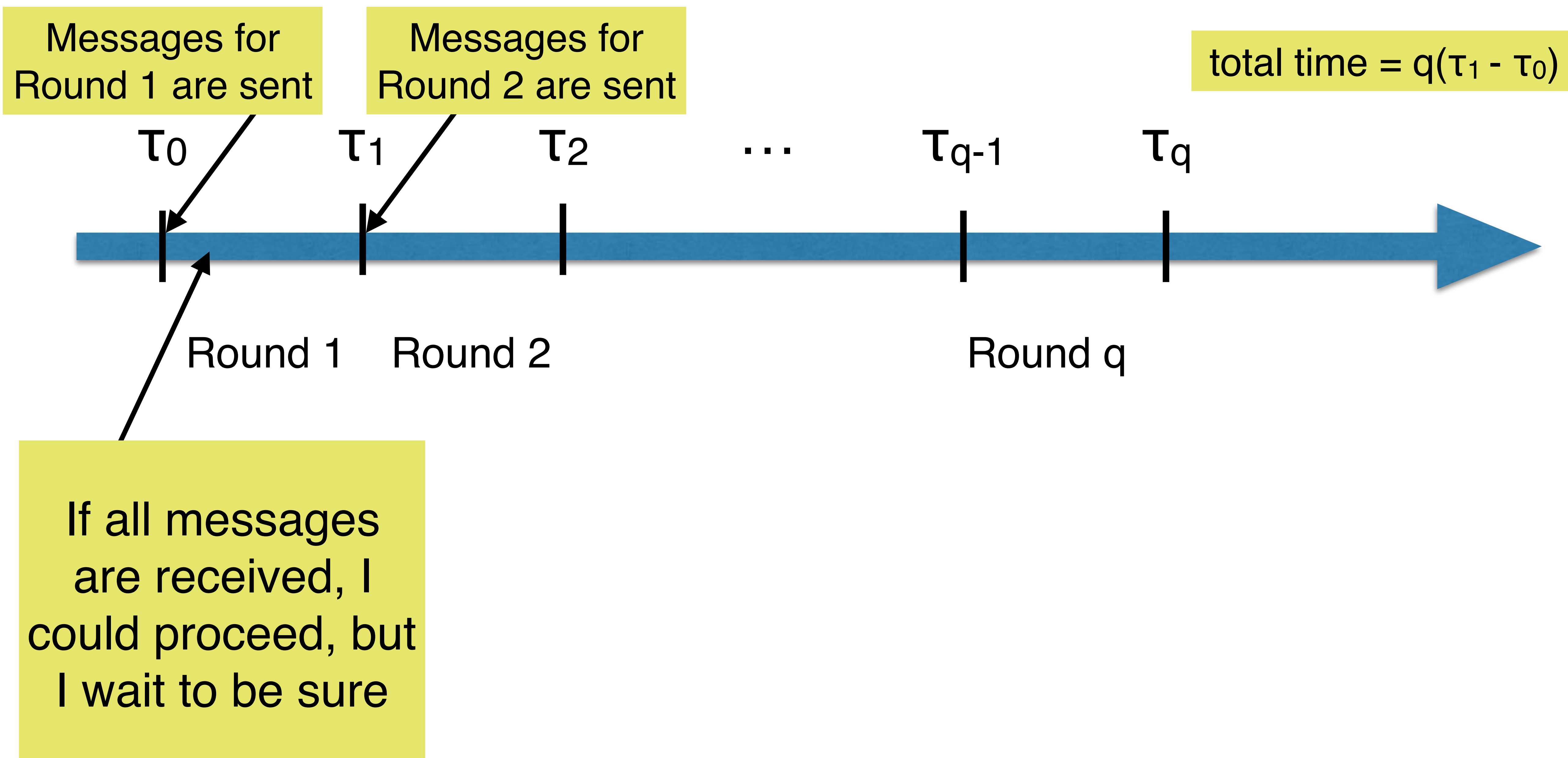
Why Asynchronous Computation?

Timeline of a Synchronous protocol



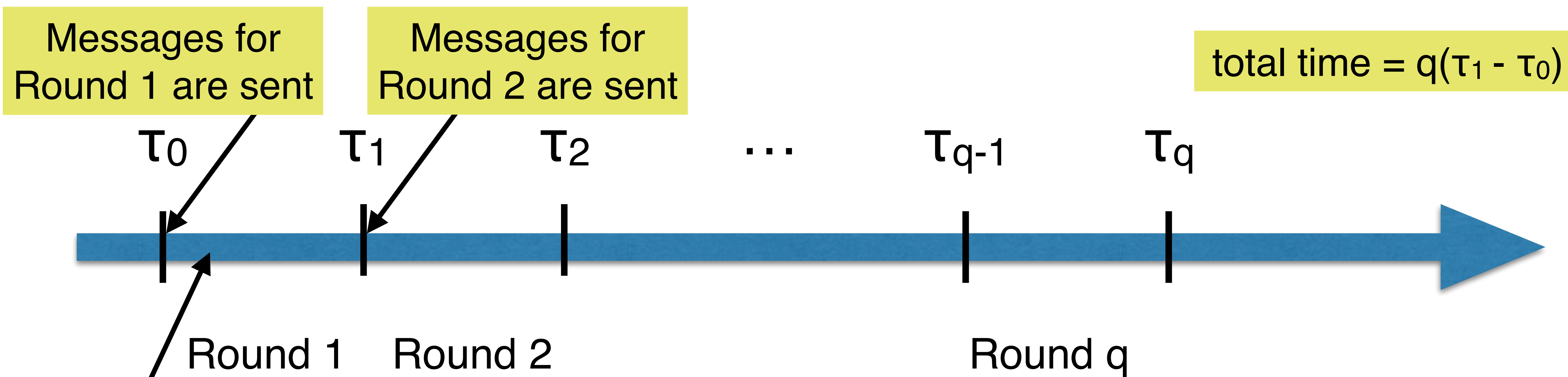
Why Asynchronous Computation?

Timeline of a Synchronous protocol



Why Asynchronous Computation?

Timeline of a Synchronous protocol



If all messages are received, I could proceed, but I wait to be sure

Asynchronous computation offers an *opportunistic/ greedy* approach to protocol execution:

- As soon as a party has enough info, he proceeds to the next round

The Asynchronous Model(s)

We want to capture a setting where the messages are delayed in the network

The Asynchronous Model(s)

We want to capture a setting where the messages are delayed in the network

Worst-case scenario:

- The delivery is the one that favors the adversary the most
- **The adversary is also the scheduler:** When a message is sent from p_i to p_j, the adversary decides if and when it will be received. Two flavors:
  1. *Fully asynchronous:* The adversary can delay messages indefinitely (This is the underlying UC network [Can00])
  2. *Asynchronous with eventual delivery:* The adversary can delay messages by a finite (polynomial) amount of time

From Synchronous to Asynchronous MPC

Outline of the lecture

- Fully asynchronous setting — Semi-honest
- Eventual-delivery setting — Semi-honest
- Fully asynchronous setting — Malicious
- Eventual delivery setting — Malicious

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**
- **Eventual-delivery setting — Semi-honest**
- **Fully asynchronous setting — Malicious**
- **Eventual delivery setting — Malicious**

Goal of this lecture: Understand the differences in the synchronous and the asynchronous model(s)

From Synchronous to Asynchronous MPC

Outline of the lecture

- Fully asynchronous setting — Semi-honest

ng — Semi-honest

tting — Malicious

ng — Malicious

Goal of this lecture: Understand the differences in the synchronous and the asynchronous model(s)

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**

ZZzzz

ng

ttin

ng

Goal of this lecture: Understand the differences in the synchronous and the asynchronous model(s)

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**
- **Eventual-delivery setting — Semi-honest**
- **Fully asynchronous setting — Malicious**
- **Eventual delivery setting — Malicious**

Goal of this lecture: Understand the differences in the synchronous and the asynchronous model(s)

From Synchronous to Asynchronous MPC

Outline of the lecture

- Fully asynchronous setting — Semi-honest
- Eventual-delivery setting — Semi-honest
- Fully asynchronous setting — Malicious
- Eventual delivery setting — Malicious

Goal of this lecture: Understand the differences in the synchronous and the asynchronous model(s)

Full Asynchrony — Semi-honest

Semi-honest **synchronous protocols can be directly executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to
- P_i: Upon receiving all messages for round ρ, compute and send your messages for round $\rho+1$

Full Asynchrony — Semi-honest

Semi-honest **synchronous protocols can be directly executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to
- P_i: Upon receiving all messages for round ρ, compute and send your messages for round $\rho+1$

Security

- No party starts round $\rho+1$ unless all parties have finished round ρ, hence the view is identical to the synchronous protocol.
- The privacy follows from the privacy of the synchronous protocol.

Full Asynchrony — Semi-honest

Semi-honest **synchronous protocols can be directly executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to
- P_i: Upon receiving all messages for round ρ, compute and send your messages for round $\rho+1$

Security

- No party starts round $\rho+1$ unless all parties have finished round ρ, hence the view is identical to the synchronous protocol.
- The privacy follows from the privacy of the synchronous protocol.

But since the adversary might delay messages indefinitely,
the protocols might not terminate!

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**
  - Same security as in the synchronous setting
- **Eventual-delivery setting — Semi-honest**
- **Fully asynchronous setting — Malicious**
- **Eventual delivery setting — Malicious**

Eventual Delivery — Semi-honest

The same idea as full asynchrony works *… and ensures (eventual) termination*

Eventual Delivery — Semi-honest

The same idea as full asynchrony works *… and ensures (eventual) termination*

- Every party appends to each message the round number it belongs to
- P_i: Upon receiving all messages for round ρ, compute and send your messages for round $\rho+1$

Eventual Delivery — Semi-honest

The same idea as full asynchrony works *… and ensures (eventual) termination*

- Every party appends to each message the round number it belongs to
- P_i: Upon receiving all messages for round ρ, compute and send your messages for round ρ+1

This is the fastest way to execute semi-honest protocols.

- In reality, TCP/IP will take care of this as it will re-send messages when no acknowledgment is received

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**
    - Same security as in the synchronous setting
- **Eventual-delivery setting — Semi-honest**
    - Same security as in the synchronous setting
- **Fully asynchronous setting — Malicious**
- **Eventual delivery setting — Malicious**

Full Asynchrony — Malicious

***Malicious* synchronous protocols can be compiled to be executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to.
- P_i: Upon receiving all messages for round ρ,
  1. Compute and send your messages for round ρ+1
  2. Send a heart-bit to every party with the current round
- Upon receiving heart-bit for round ρ from every party proceed to round ρ+1

Full Asynchrony — Malicious

***Malicious* synchronous protocols can be compiled to be executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to.
- P_i: Upon receiving all messages for round ρ,
  1. Compute and send your messages for round $\rho+1$
  2. Send a heart-bit to every party with the current round
- Upon receiving heart-bit for round ρ from every party proceed to round $\rho+1$

Security

- No party starts round $\rho+1$ unless all parties have finished round ρ, hence the view is identical to the synchronous protocol.
- Privacy and correctness follow from the privacy and correctness of the synchronous protocol.

Full Asynchrony — Malicious

Malicious **synchronous protocols can be compiled to be executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to.
- P_i: Upon receiving all messages for round ρ,
    1. Compute and send your messages for round $\rho+1$
    2. Send a heart-bit to every party with the current round
- Upon receiving heart-bit for round ρ from every party proceed to round $\rho+1$

But the adversary can prevent the protocol from terminating

Security

- No party starts round $\rho+1$ unless all parties have finished round ρ, hence the view is identical to the synchronous protocol.
- Privacy and correctness follow from the privacy and correctness of the synchronous protocol.

Full Asynchrony — Malicious

***Malicious* synchronous protocols can be compiles to be executed on an asynchronous network:**

- Every party appends to each message the round number it belongs to.
- P_i: Upon receiving all messages for round ρ,
  1. Compute and send your messages for round ρ+1
  2. Send a heart-bit to every party with the current round
- Upon receiving heart-bit for round ρ from every party proceed to round ρ+1

But the adversary can prevent the protocol from terminating

Security *without* termination is infeasible in the *fully* asynchronous model

- The adversary can make sure that no message is ever delivered

From Synchronous to Asynchronous MPC

Outline of the lecture

- **Fully asynchronous setting — Semi-honest**
  - Same security as in the synchronous setting
- **Eventual-delivery setting — Semi-honest**
  - Same security as in the synchronous setting
- **Fully asynchronous setting — Malicious**
  - Same security as in the synchronous setting … but no termination
- **Eventual delivery setting — Malicious**

Eventual Delivery— Malicious

If you don’t care about termination then trivial: use the fully asynchronous protocol idea…

Eventual Delivery— Malicious

If you don’t care about termination then trivial: use the fully asynchronous protocol idea…

… could we get (eventual) termination as in the semi-honest setting ?

Eventual Delivery— Malicious

If you don’t care about termination then trivial: use the fully asynchronous protocol idea…

… could we get (eventual) termination as in the semi-honest setting ?

Yes !!! …

Eventual Delivery— Malicious

If you don't care about termination then trivial: use the fully asynchronous protocol idea…

… could we get (eventual) termination as in the semi-honest setting ?

Yes !!! …

… but at a cost …

Eventual Delivery— Fail-stop

A ***fail-stop*** adversary might make corrupted parties *crash,* i.e., stop playing but cannot make them misbehave in other ways.

A ***fail-stop*** adversary is strictly weaker than a malicious adversary so any limitations transfer to the malicious model.

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

(Recall:) **Broadcast**

Inputs: A party p_i called *the sender* has input x

Outputs: Every p_j outputs y_j

- (consistency) There exists y s.t. y_j = y for all j
- (validity) If p_i is honest (i.e., does not crash) then y = x
- (termination) The protocol eventually terminates

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

Synchronous broadcast against fail-stop sender:

- Round 1: Sender sends his input x to every p_i
- Round 2: Every p_i sends the message he received (or $\perp$ if no message was received) to all p_j ’s
- Output: For each p_i : if a message $x \neq \perp$ was received in Round 1 or 2 output x otherwise output $\perp$.

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

Synchronous broadcast against fail-stop sender:

- Round 1: Sender sends his input x to every p_i
- Round 2: Every p_i sends the message he received (or $\perp$ if no message was received) to all p_j ’s
- Output: For each p_i : if a message $x \neq \perp$ was received in Round 1 or 2 output x otherwise output $\perp$.

Security:

- Consistency:
  - If any party receives a message $x \neq \perp$ in Round 1 then everyone will output x in Round 2. Otherwise everyone output $\perp$.
- Validity: If the Sender is honest everyone receives x already in Round 1 (and output it in the end).

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about asynchronous broadcast against fail-stop sender

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about asynchronous broadcast against fail-stop sender

- If the parties do not wait for the sender then they might compromise validity
  - The sender might be honest but his network very slow …
- Hence the parties need to wait for the sender
  - But then a fail-stop sender will make them wait forever …

Eventual Delivery — Fail-stop

The "simple" case of Broadcast

How about asynchronous broadcast against fail-stop sender

- If the parties do not wait for the sender then they might compromise validity
  - The sender might be honest but his network very slow …
- Hence the parties need to wait for the sender
  - But then a fail-stop sender will make them wait forever …

Theorem [FLP85]. Broadcast with eventual (guaranteed) termination is impossible in the eventual-delivery asynchronous setting if the sender is semi-honest (or malicious).

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about asynchronous broadcast against fail-stop sender

Let’s try anyway to use the idea of the synchronous protocol:

- Start (Round 1): Sender sends his input x to every p_i
- Every p_i who receives some x from the sender or some p_j echoes x and terminates with output x.

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about asynchronous broadcast against fail-stop sender

Let’s try anyway to use the idea of the synchronous protocol:

- Start (Round 1): Sender sends his input x to every p_i
- Every p_i who receives some x from the sender or some p_j echoes x and terminates with output x.

“Asynchronous” Broadcast (aka Bracha broadcast [Bra84]**)**

- (validity) If the sender is honest with input x then every party eventually terminates with output x
- (conditional consistency) If some honest party terminates with x’ then every honest party will (eventually) terminate with x’.

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about asynchronous broadcast against fail-stop sender

Let’s try anyway to use the idea of the synchronous protocol:

- Start (Round 1): Sender sends his input x to every p_i
- Every p_i who receives some x from the sender or some p_j echoes x and terminates with output x.

Tolerates up to t<n/3 *malicious* parties

“Asynchronous” Broadcast (aka Bracha broadcast [Bra84]**)**

- (validity) If the sender is honest with input x then every party eventually terminates with output x
- (conditional consistency) If some honest party terminates with x’ then every honest party will (eventually) terminate with x’.

Eventual Delivery— Fail-stop

The “simple” case of Broadcast

How about MPC?

How about asynchronous broadcast against fail-stop sender

Let’s try anyway to use the idea of the synchronous protocol:

- Start (Round 1): Sender sends his input x to every p_i
- Every p_i who receives some x from the sender or some p_j echoes x and terminates with output x.

Tolerates up to $t<n/3$ *malicious* parties

“Asynchronous” Broadcast (aka Bracha broadcast [Bra84]**)**

- (validity) If the sender is honest with input x then every party eventually terminates with output x
- (conditional consistency) If some honest party terminates with x’ then every honest party will (eventually) terminate with x’.

Eventual Delivery— Malicious

The case of general MPC: If correctness requires receiving input from all honest parties then they will *not* terminate even against a single corruption

- If the parties do not wait for some p_i 's input then they might compromise correctness
  - p_i might be honest but his network very slow …
- Hence the parties need to wait for p_i
  - But then a malicious (or fail-stop) p_i will make them wait forever …

Eventual Delivery— Malicious

The case of general MPC: If correctness requires receiving input from *all but one* honest parties then they will not terminate against two corruption

- Assume the parties give up waiting for p_i 's input (no correctness violation)
- If the parties do not wait for some p_j 's input then they might compromise correctness
  - p_j might be honest but his network very slow …
- Hence the parties need to wait for p_j
  - But then a malicious (or fail-stop) p_j will make them wait forever …

Eventual Delivery— Malicious

The case of general MPC: If correctness requires receiving input from *all but t-1* honest parties then they will not terminate against t corruption

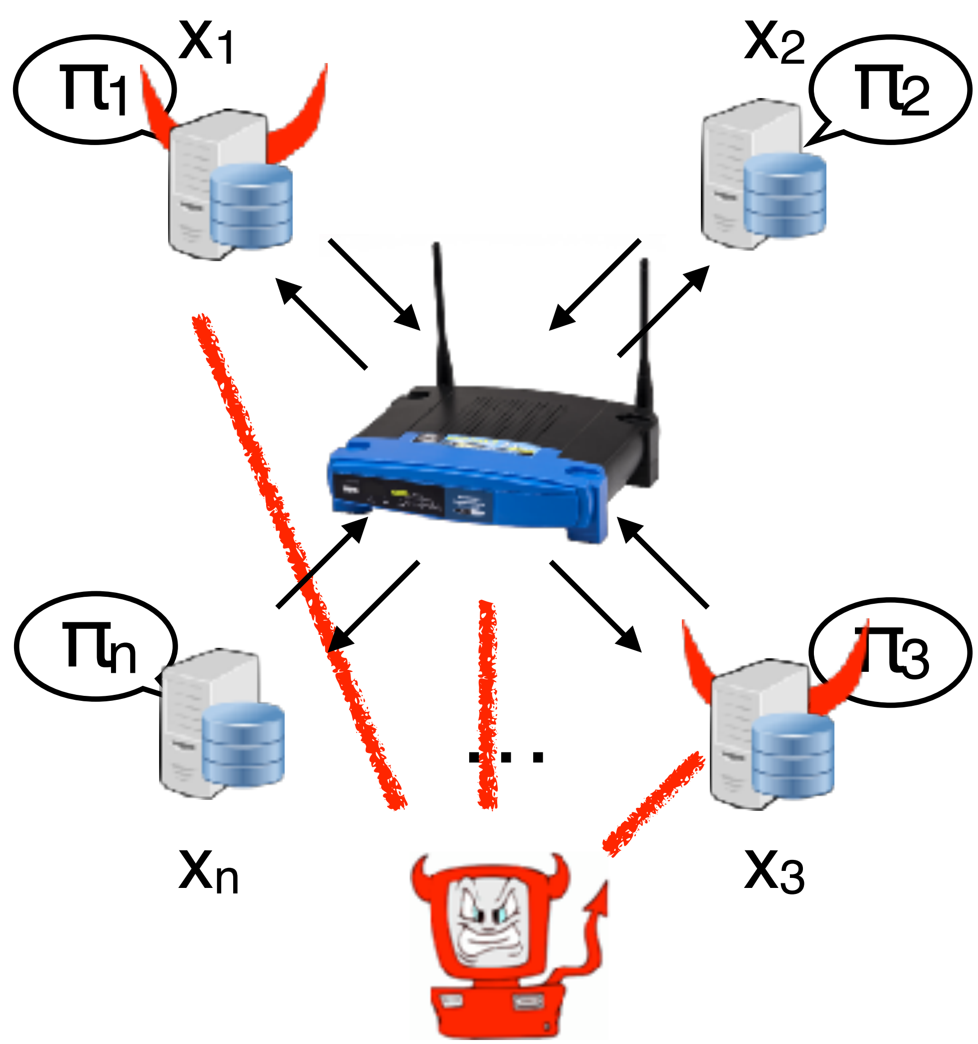
Eventual Delivery— Malicious

The case of general MPC: If correctness requires receiving input from *all but t-1* honest parties then they will not terminate against t corruption

The best we can hope for is that parties give up t honest parties in correctness.

MPC Security — Synchronous Model

Protocol for $f(x_1, \dots, x_n)$

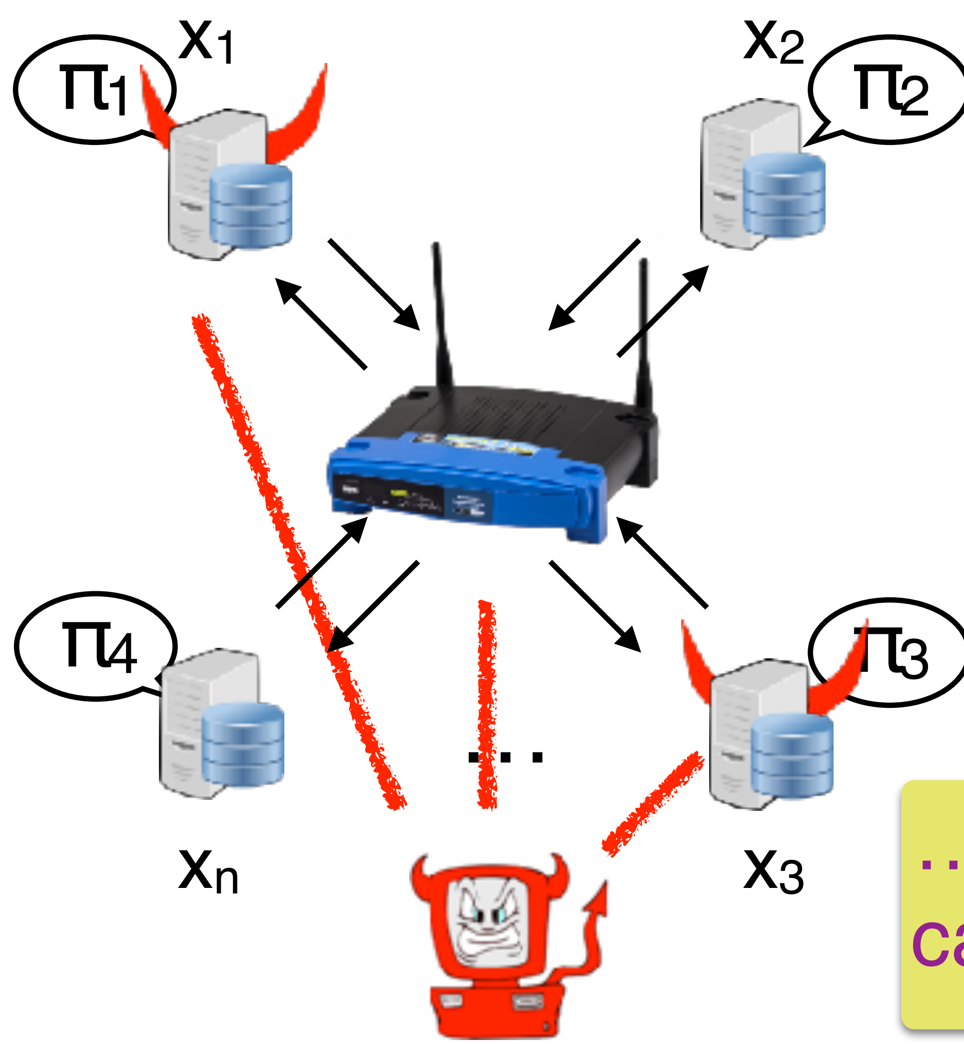


Protocol π is secure if for every adversary:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties output $f(x_1', x_2, x_3', \dots, x_n)$
- *(termination)* The protocol terminates after a finite number of rounds

MPC Security — Eventual Delivery Model

Protocol for $f(x_1, \ldots, x_n)$



… where the adversary can set t honest x_i 's to 0

Protocol π is secure if for every adversary:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties output $f(x_1', x_2, x_3', \ldots, x_n)$
- *(eventual termination)* The protocol eventually terminates

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

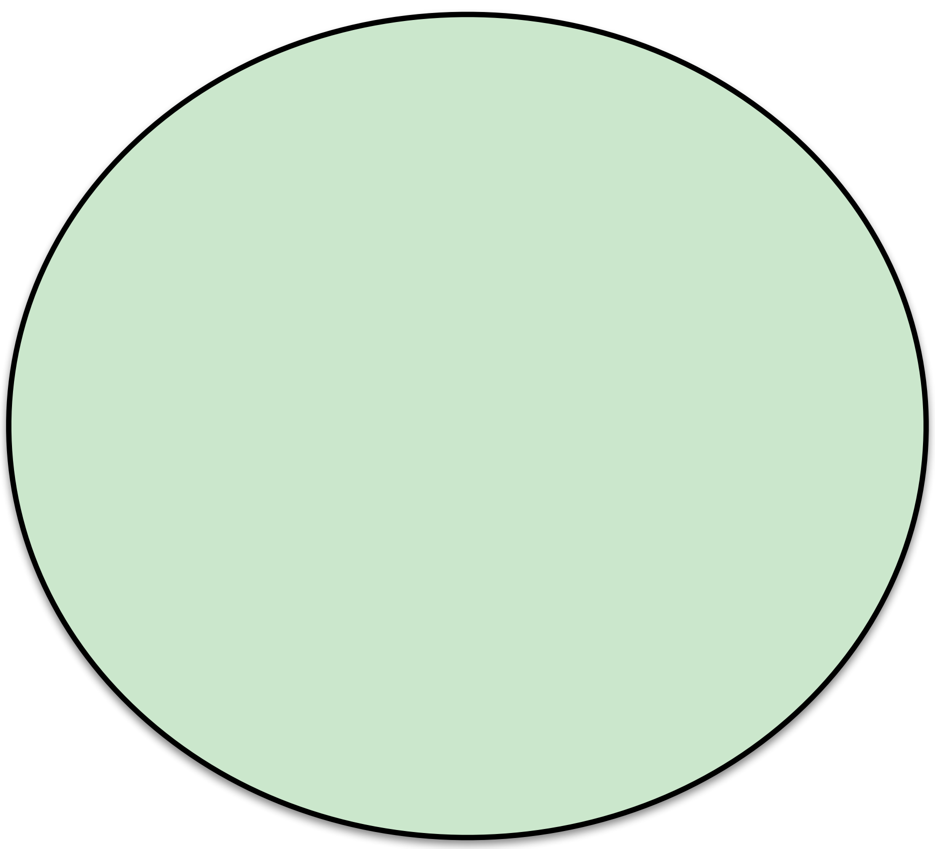
A. Unfortunately not …

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



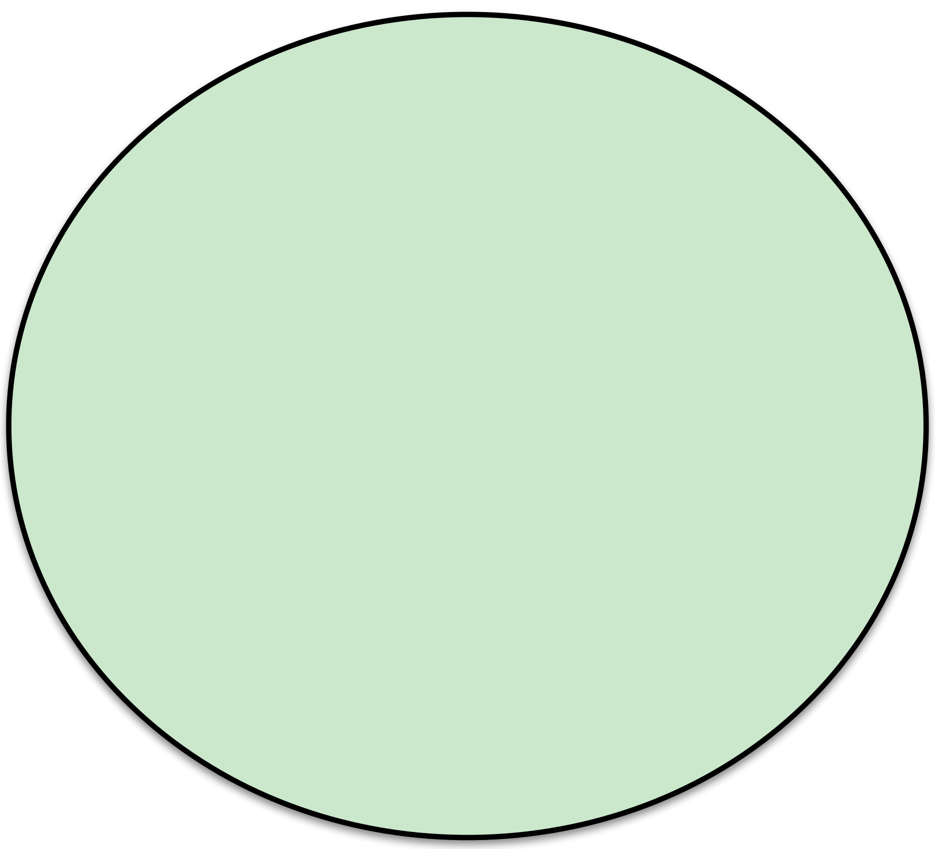
MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

- No party can wait for messages from more than n-t parties

Player set $\{p_1, \ldots, p_n\}$

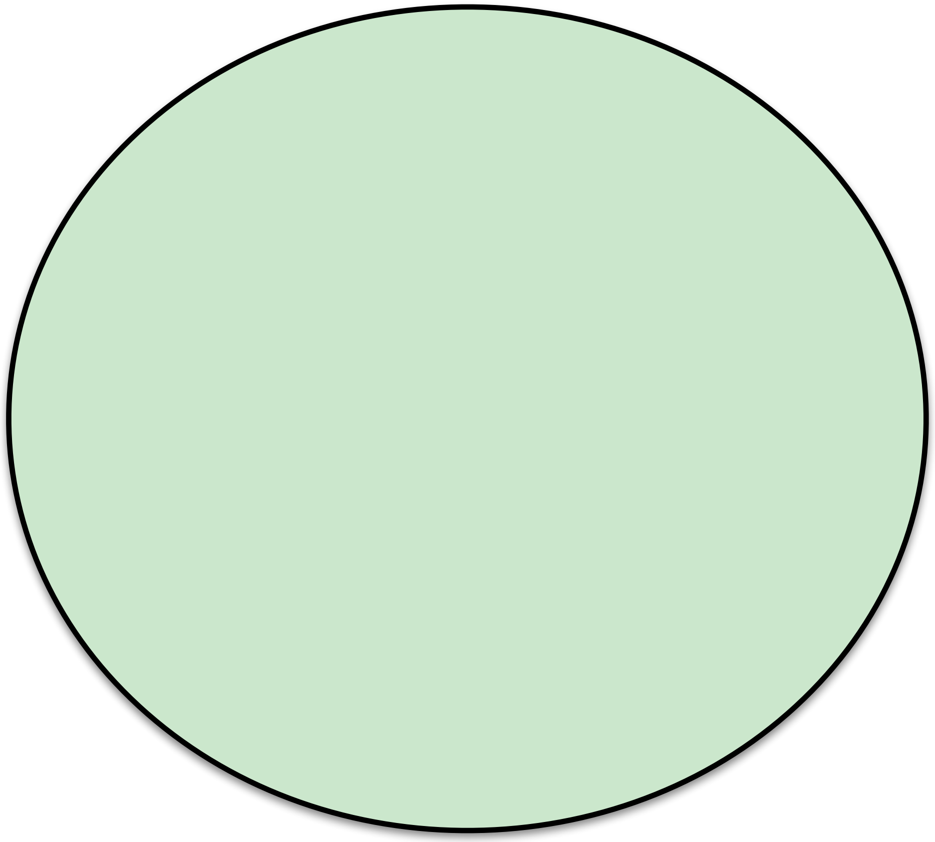


MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



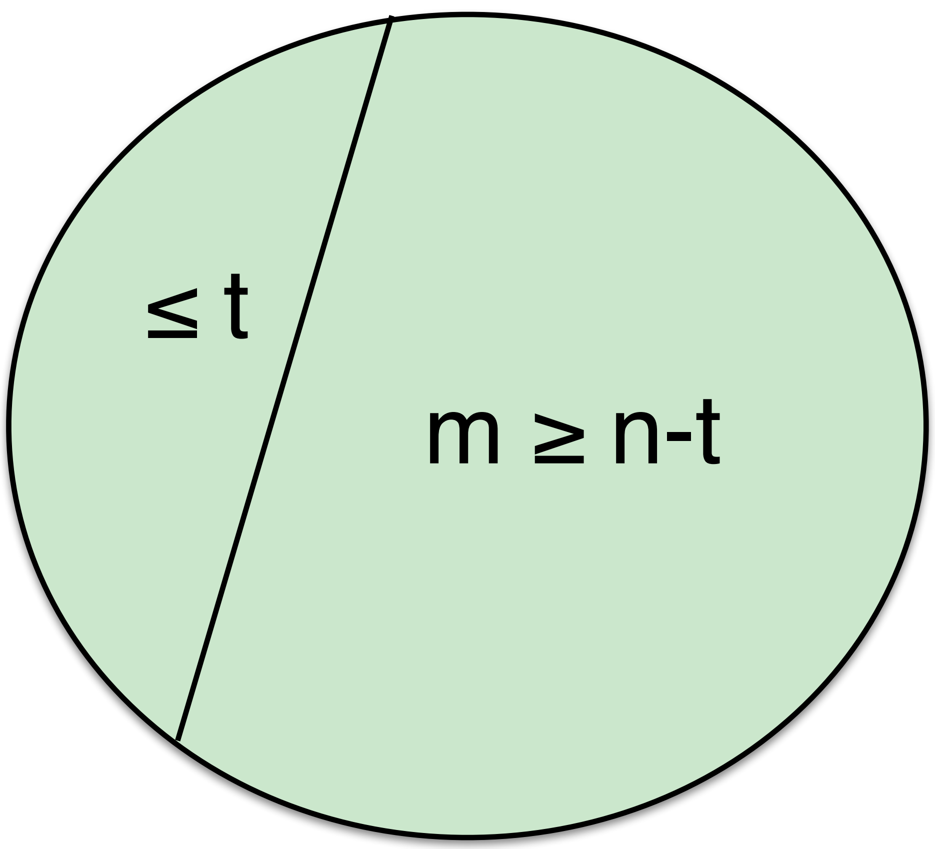
- No party can wait for messages from more than n-t parties
- The adversary chooses who is left behind (by delaying delivery)
  - Best strategy: leave out t *honest* parties

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



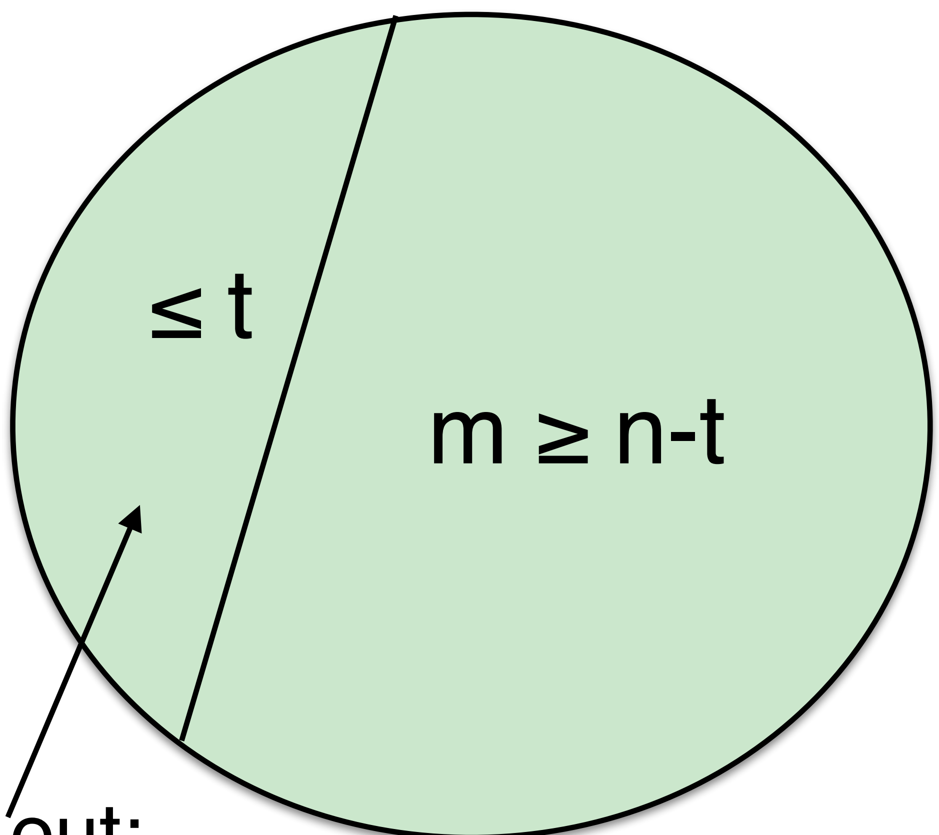
- No party can wait for messages from more than n-t parties
- The adversary chooses who is left behind (by delaying delivery)
  - Best strategy: leave out t *honest* parties

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



Left out:
Might be all
honest

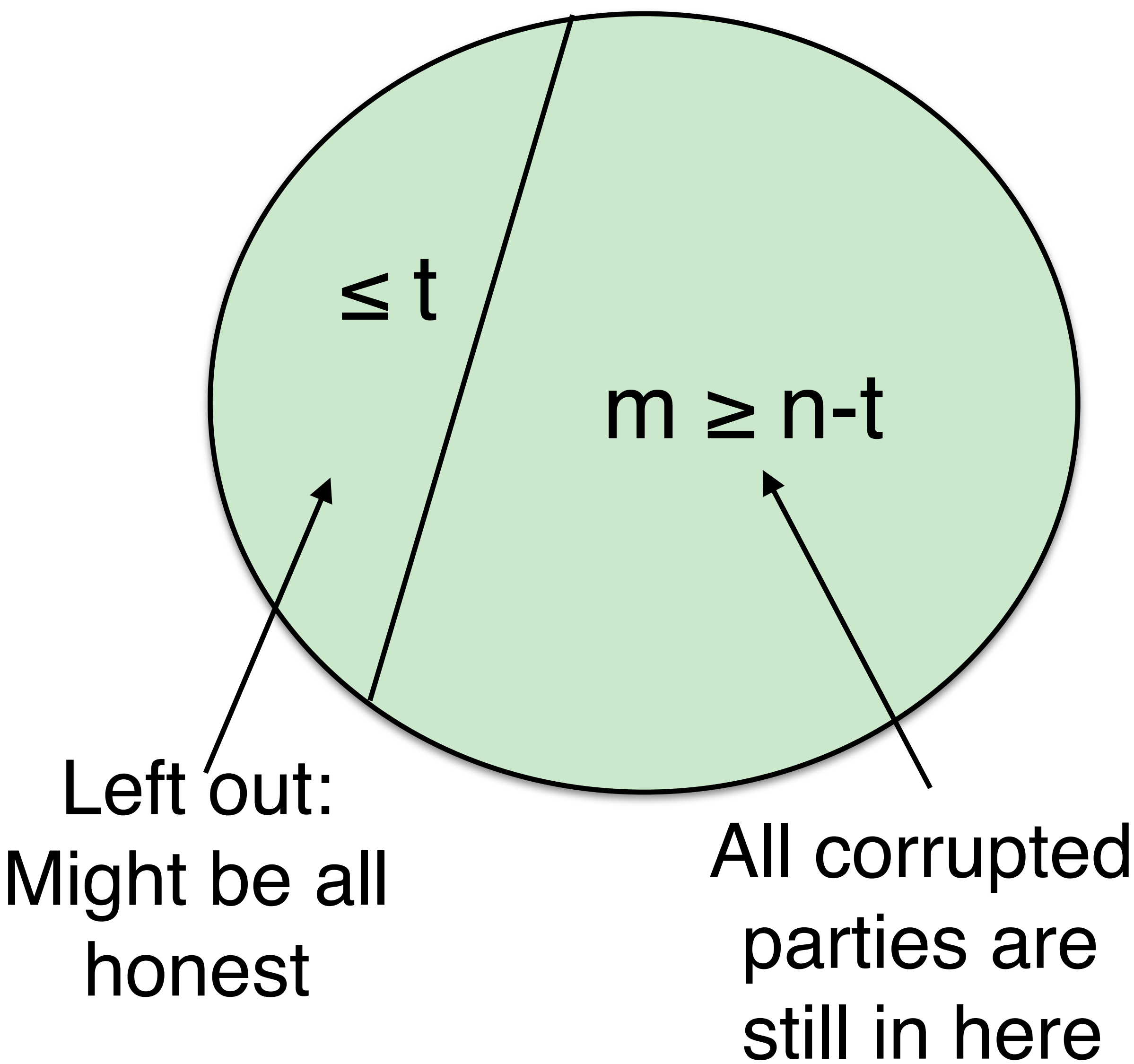
- No party can wait for messages from more than n-t parties
- The adversary chooses who is left behind (by delaying delivery)
  - Best strategy: leave out t *honest* parties

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



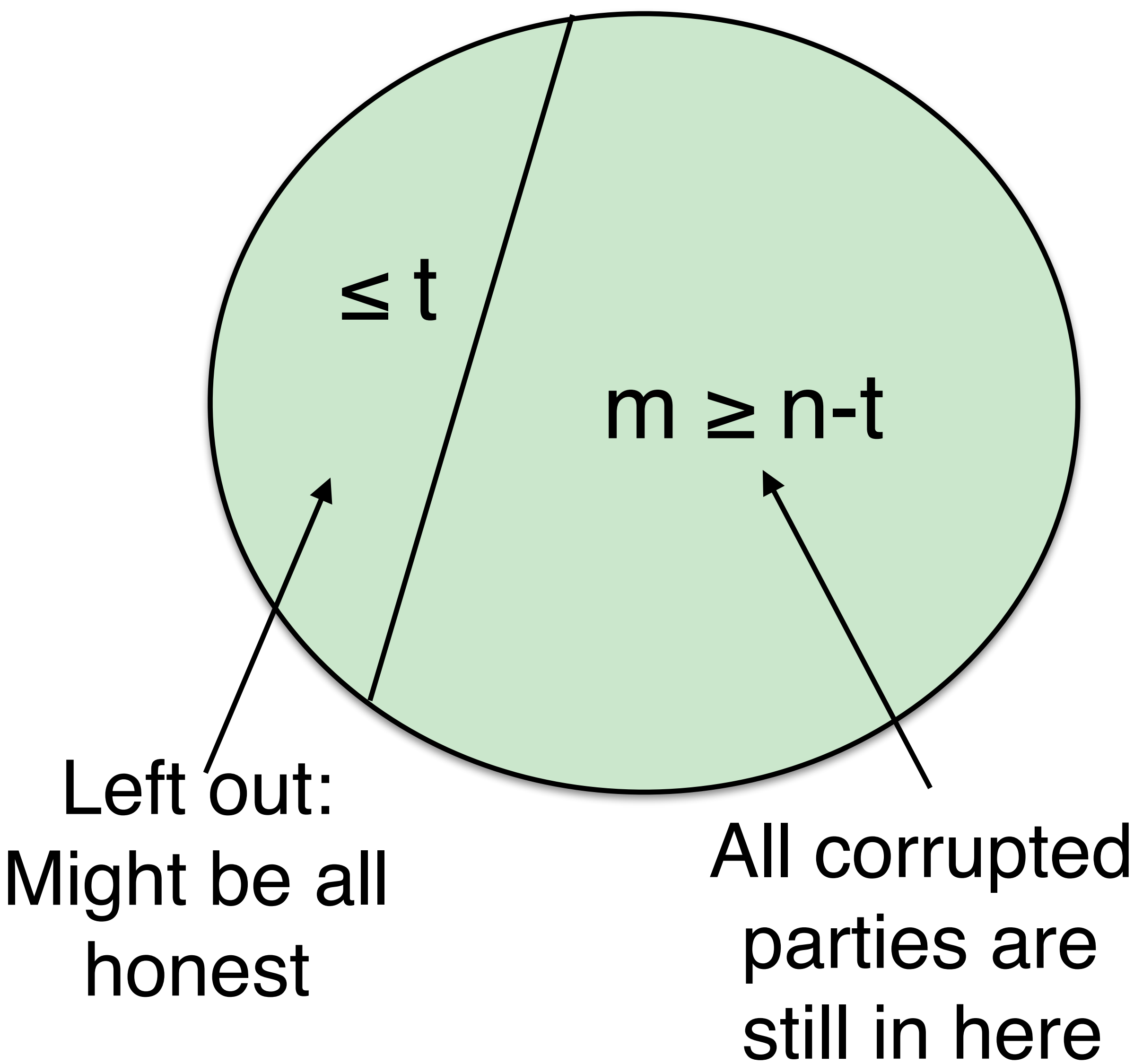
- No party can wait for messages from more than n-t parties
- The adversary chooses who is left behind (by delaying delivery)
  - Best strategy: leave out t *honest* parties

MPC Security — Eventual Delivery

Q. Can we achieve the synchronous feasibility bounds?

A. Unfortunately not …

Player set $\{p_1, \ldots, p_n\}$



- No party can wait for messages from more than n-t parties
- The adversary chooses who is left behind (by delaying delivery)
  - Best strategy: leave out t *honest* parties
- Even if the adversary synchronously delivers all messages in the m ≥ n-t remainder parties … we need to pay the synchronous penalties:
  - (perfect) $m > 3t \Rightarrow n > 4t$ [BCG93]
  - (computational/IT) $m > 2t \Rightarrow n > 3t$ [BKR94]

MPC Security — Eventual Delivery

(Over-simplified) Idea of asynchronous protocols

The most important component is a primitive called *core-set agreement (CSA)* *[BCG93, BKR94]*

- *Allows the parties to (eventually) agree on a core-set of n-t parties who have completed their previous step (typically sharing of their input).*

MPC Security — Eventual Delivery

(Over-simplified) Idea of asynchronous protocols

The most important component is a primitive called *core-set agreement (CSA)* *[BCG93, BKR94]*

- *Allows the parties to (eventually) agree on a core-set of n-t parties who have completed their previous step (typically sharing of their input).*

$\Downarrow$

Asynchronous VSS:

- Every party verifiably shares his inputs
- Run core-set agreement to decide on n-t parties who have successfully VSS-ed their inputs.

MPC Security — Eventual Delivery

(Over-simplified) Idea of asynchronous protocols

The most important component is a primitive called *core-set agreement (CSA)* *[BCG93, BKR94]*

- *Allows the parties to (eventually) agree on a core-set of n-t parties who have completed their previous step (typically sharing of their input).*

$\Downarrow$

Asynchronous VSS:

- Every party verifiably shares his inputs
- Run core-set agreement to decide on n-t parties who have successfully VSS-ed their inputs.

Given these primitives, the structure is similar to the synchronous protocols: parties use CSA to detect that the evaluation of a gate has finished and they can proceed to the next gate.

MPC Security — Eventual Delivery

(Over-simplified) Idea of asynchronous protocols

The most important component is a primitive called *core-set agreement (CSA)* *[BCG93, BKR94]*

- *Allows the parties to (eventually) agree on a core-set of n-t parties who have completed their previous step (typically sharing of their input).*

$\Downarrow$

Asynchronous VSS:

- Every party verifiably shares his inputs
- Run core-set agreement to decide on n-t parties who have successfully VSS-ed their inputs.

Given these primitives, the structure is similar to the synchronous protocols: parties use CSA to detect that the evaluation of a gate has finished and they can proceed to the next gate.

Detailed analysis is involved:

- Complications + reduced correctness = not a lot of literature

MPC Security — Eventual Delivery

MPC Security — Eventual Delivery

Why is should we look at asynchronous with eventual delivery?

MPC Security — Eventual Delivery

Why is should we look at asynchronous with eventual delivery?

- Because we cannot always assume that parties have synchronized clocks.
  - What can we do if not?

MPC Security — Eventual Delivery

Why is should we look at asynchronous with eventual delivery?

- Because we cannot always assume that parties have synchronized clocks.
  - What can we do if not?
- Because it is an interesting theoretical problem.

MPC Security — Eventual Delivery

Why is should we look at asynchronous with eventual delivery?

- Because we cannot always assume that parties have synchronized clocks.
  - What can we do if not?
- Because it is an interesting theoretical problem.
- Because we might only be able to have a pessimistic guarantee on the network delay.
  - Synchronous protocols will be too slow.
  - We could get results in a hybrid (optimistic model):
    - synchronous with asynchronous fallback

MPC Security — Eventual Delivery

Why is should we look at asynchronous with eventual delivery?

- Because we cannot always assume that parties have synchronized clocks.
  - What can we do if not?
- Because it is an interesting theoretical problem.
- Because we might only be able to have a pessimistic guarantee on the network delay.
  - Synchronous protocols will be too slow.
  - We could get results in a hybrid (optimistic model):
    - synchronous with asynchronous fallback

MPC Security — Eventual Delivery

MPC Security — Eventual Delivery

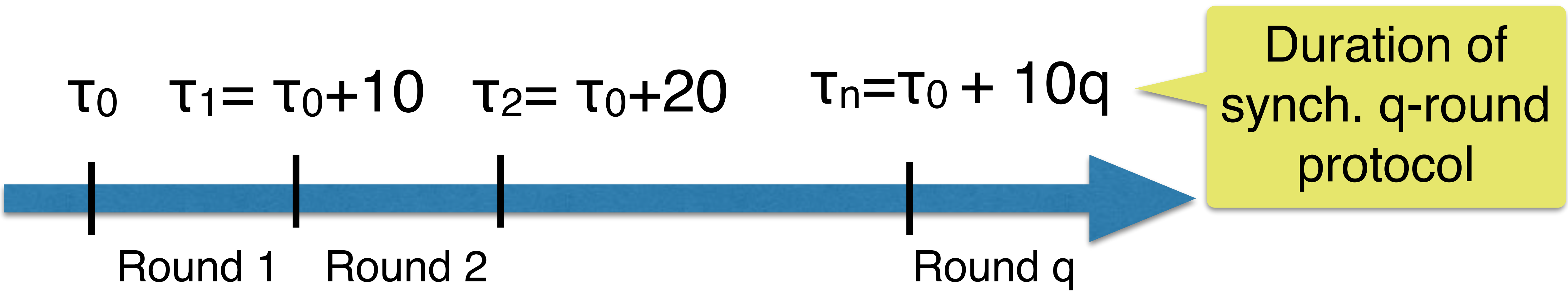
A optimistic protocol without correctness compromise:

- Assume we know that messages are almost never delayed more than 10mins, but *typically* they are delivered in 1sec.

MPC Security — Eventual Delivery

A optimistic protocol without correctness compromise:

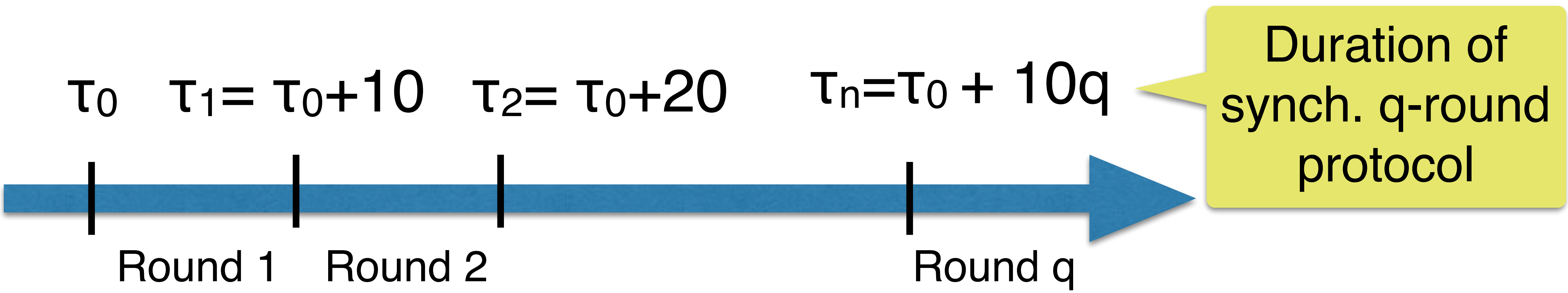
- Assume we know that messages are almost never delayed more than 10mins, but *typically* they are delivered in 1sec.
- In a synchronous protocol I would need #rounds · 10mins time …



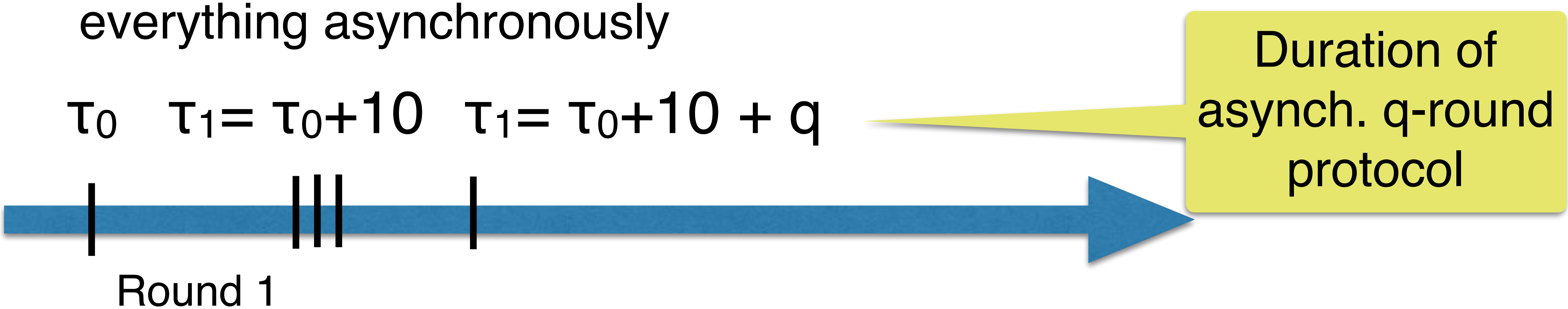
MPC Security — Eventual Delivery

A optimistic protocol without correctness compromise:

- Assume we know that messages are almost never delayed more than 10mins, but *typically* they are delivered in 1sec.
- In a synchronous protocol I would need #rounds · 10mins time …



- A better idea: Run the first round for 10 mins and then do everything asynchronously



MPC Security — Eventual Delivery

A optimistic protocol without correctness compromise:

Theorem. [HNP05, BH07] Assuming the messages send at the beginning of the protocol are delivered to their recipients synchronously (within the first 10 mins), we can achieve the same correctness as in the synchronous setting (i.e, compute the function on all the inputs) faster but under the asynchronous bounds.

- perfect security: $n > 4t$
- (computational/IT): $n > 3t$

MPC Security — Eventual Delivery

A optimistic protocol without correctness compromise:

Theorem. [HNP05, BH07] Assuming the messages send at the beginning of the protocol are delivered to their recipients synchronously (within the first 10 mins), we can achieve the same correctness as in the synchronous setting (i.e, compute the function on all the inputs) faster but under the asynchronous bounds.

- perfect security: $n > 4t$
- (computational/IT): $n > 3t$

MPC Security — Eventual Delivery

MPC Security — Eventual Delivery

A protocol for a function $f(x_1, \ldots, x_n)$ with *full correctness* for $t<n/3$ (assuming digital signatures)

MPC Security — Eventual Delivery

A protocol for a function $f(x_1, \ldots, x_n)$ with *full correctness* for $t<n/3$ (assuming digital signatures)

1. Protocol start (*synchronous round*):
   - Every party p_i computes a sharing of his input x_i using a degree-t polynomial $f_i(\cdot)$.
   - p_i send $x_{ij}=f(\alpha_j)$ and his signature $\sigma_{ij} = sig_{sk_i}(x_{ij},ij)$ to each p_j.

MPC Security — Eventual Delivery

A protocol for a function $f(x_1, \ldots, x_n)$ with *full correctness* for $t<n/3$ (assuming digital signatures)

1. Protocol start (*synchronous round*):
   - Every party p_i computes a sharing of his input x_i using a degree-t polynomial $f_i(\cdot)$.
   - p_i send $x_{ij}=f(\alpha_j)$ and his signature $\sigma_{ij} = sig_{sk_i}(x_{ij}, ij)$ to each p_j.
2. The parties use an asynchronous protocol for $t<n/3$ (e.g., [BKR94]) to compute the following function on input the shares and signatures received in the first round:

MPC Security — Eventual Delivery

A protocol for a function $f(x_1, \ldots, x_n)$ with *full correctness* for $t<n/3$ (assuming digital signatures)

1. Protocol start (*synchronous round*):
   - Every party p_i computes a sharing of his input x_i using a degree-t polynomial $f_i(\cdot)$.
   - p_i send $x_{ij}=f(\alpha_j)$ and his signature $\sigma_{ij} = sig_{sk_i}(x_{ij},ij)$ to each p_j.
2. The parties use an asynchronous protocol for $t<n/3$ (e.g., [BKR94]) to compute the following function on input the shares and signatures received in the first round:

$G((x_{11}, \sigma_{11}), \ldots, (x_{nn}, \sigma_{nn}))$: For all received inputs (x_{ij}, σ_{ij}) with a valid signature:

- For each $i \in \{1, \ldots, n\}$:
  - If there exists a degree-t polynomial $g_i(\cdot)$ such that $g_i(\alpha_j) = x_{ij}$ then set $x_i' = g(0)$
  - Else set $x_i' = 0$ (a default value)
- Compute $f(x_1, \ldots, x_n)$

MPC Security — Eventual Delivery

A protocol for a function $f(x_1, \ldots, x_n)$ with full correctness for $t<n/3$ (assuming digital signatures)

Security Proof for $t<n/3$

Correctness: If p_i is honest then his input x_i is considered in the evaluation

- In the synchronous round everyone receives his share and signature (s_{ij}, σ_{ij})
- Even if the evaluation of G leaves t honest parties behind there is t+1 more honest that have shares to interpolate the polynom. f_i

Privacy & Termination: Follow from the asynch. protocol used for G.

$G((x_{11}, \sigma_{11}), \ldots, (x_{nn}, \sigma_{nn}))$: For all received inputs (x_{ij}, σ_{ij}) with a valid signature:

- For each $i \in \{1, \ldots, n\}$:
  - If there exists a degree-t polynomial $g_i(\cdot)$ such that $g_i(\alpha_j) = x_{ij}$ then set $x_i' = g(0)$
  - Else set $x_i' = 0$ (a default value)
- Compute $f(x_1, \ldots, x_n)$

MPC Security — Eventual Delivery

MPC Security — Eventual Delivery

Theorem *(informal)***.** [HNP05, BH07] *Best of both worlds:*

Under the asynchronous bounds we can have a protocol with delay (due to time-outs) almost τ which computes any multi-party function $f(x_1,\ldots,x_n)$ s.t.,

Correctness:

- If the inputs are received within time τ (i.e., by the end of first round) then full correctness (as above)
- Else, still correctness which leaves out at most t honest inputs

Privacy & Eventual Termination:

- Guaranteed irrespective of synchrony

MPC Security — Eventual Delivery

Theorem *(informal)*. [HNP05, BH07] *Best of both worlds:*

Under the asynchronous bounds we can have a protocol with delay (due to time-outs) almost τ which computes any multi-party function $f(x_1,\ldots,x_n)$ s.t.,

Correctness:

- If the inputs are received within time τ (i.e., by the end of first round) then full correctness (as above)
- Else, still correctness which leaves out at most t honest inputs

Privacy & Eventual Termination:

- Guaranteed irrespective of synchrony

This motivates the study of practical async. MPC protocols

- Communication efficient [HNP08, CHP13, CBP15, …]
- Constant round [CGHZ16, Coh16]

References

- [Bra84] Gabriel Bracha. An asynchronous ⌊(n – 1)/3⌋-resilient con- sensus protocol. In Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), pages 154–162, 1984. P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer Science Research*, pages 313–322, 1992. Preliminary version in STOC'89.
- [FLP85] M. Fisher, N. Lynch, M. Paterson. Impossibility of Distributed Consensus with one faulty process. JACM, Vol. 32, No. 2, 1985, pp. 374—382
- [BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proc. 25th ACM Symposium on the Theory of Computing (STOC)*, pages 52–61, 1993. Full version in Ran Canetti's PhD Thesis
- [BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proc. 13th ACM Symposium on Principles of Dis- tributed Computing (PODC)*, pages 183–192, 1994.
- [HNP05] M. Hirt, J. Buus Nielsen, and B. Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer-Verlag, May 2005.
- [BH07] Z. Beerliova´-Trub´ıniova´ and M. Hirt. Simple and efficient perfectly-secure asynchronous MPC. In Kaoru Kuro- sawa, editor, *Advances in Cryptology — ASIACRYPT 2007*, vol- ume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer-Verlag, December 2007.

References

- [HNP08] M. Hirt, J. Buus Nielsen, and B. Przydatek. Asynchronous multi-party computation with quadratic com- munication. In Luca Aceto, Magnus M. Halldorsson, and Anna Ingolfsdottir, editors, *Automata, Languages and Program- ming — ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485. Springer-Verlag, July 2008.
- [CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. 2013. Asynchronous Multiparty Computation with Linear Communication Complexity. In *Proceedings of the 27th International Symposium on Distributed Computing - Volume 8205* (DISC 2013), Yehuda Afek (Ed.), Vol. 8205. Springer-Verlag New York, Inc., New York, NY, USA, 388-402. DOI=http://dx.doi.org/10.1007/978-3-642-41527-2_27
- [CB15] Ashish Choudhury and Arpita Patra. 2015. Optimally Resilient Asynchronous MPC with Linear Communication Complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking* (ICDCN '15). ACM, New York, NY, USA, Article 5, 10 pages. DOI: https://doi.org/10.1145/2684464.2684470
- [Coh16] R. Cohen. Asynchronous secure multiparty computation in constant time. In: Public-Key Cryptography - PKC 2016, Proceedings, Part II. pp. 183–207, 2016.
- [CGHZ16] S. Coretti, J. A. Garay, M. Hirt, and V. Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In J. H. Cheon and T. Takagi, editors, ASIACRYPT 2016, volume 10032 of LNCS, pages 998–1021, 2016.

Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions

S. Coretti, J. Garay, M. Hirt and V. Zikas, "Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions." ASIACRYPT 2016.

http://eprint.iacr.org/2016/208

Constant-Round Asynchronous MPC

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Prior Work Constant-Round MPC Protocols

Prior Work Constant-Round MPC Protocols

- Synchronous model:
  - Based on circuit garbling [Yao86, BMR90, DI05, IPS08]
  - Based on FHE [AJLTVW12]
  - $t < n/2$ corruptions
  - Assume broadcast channel (cf. [FL82, BE03, CCGZ16])

Prior Work Constant-Round MPC Protocols

- Synchronous model:
  - Based on circuit garbling [Yao86, BMR90, DI05, IPS08]
  - Based on FHE [AJLTVW12]
  - $t < n/2$ corruptions
  - Assume broadcast channel (cf. [FL82, BE03, CCGZ16])
- Asynchronous model (recall: eventual delivery):
  - Based on FHE [Coh16]
  - $t < n/3$ corruptions
  - Assume A-BA
  - (Other known protocols are GMW-based → circuit depth)

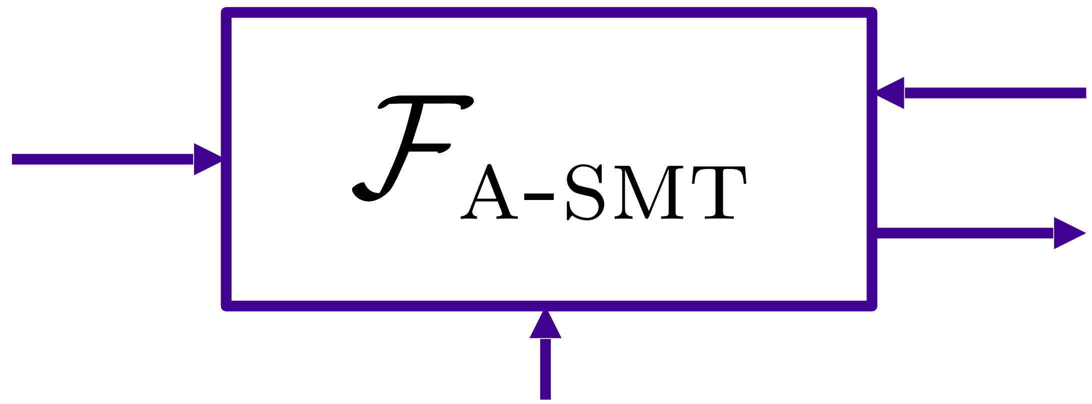
Our Results

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Modeling Asynchronous Communication in UC

Sender

Input messages



Receiver

- Poll for messages: $T = T-1$
- If $T = 0$, first message in buffer output

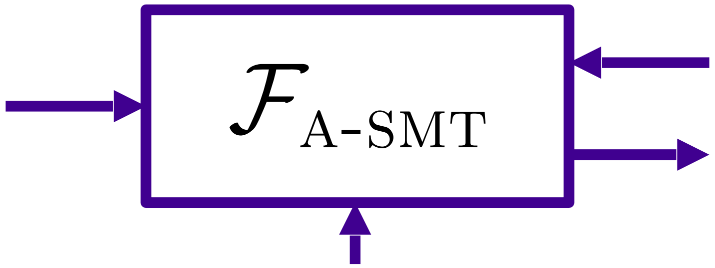
A-SMT Functionality:

- Stores messages in buffer
- Maintains delay T

Adversary

- Reorder messages in buffer
- Increase T, specified in **unary**

Modeling Asynchronous Communication in UC (2)



- Protocol execution:
  - Party either sends message or
  - polls A-SMT channels in round-robin fashion
- Round complexity: Maximum number of times any party switches between sending and polling

Modeling Asynchronous SFE in UC

Parties P

- Provide input
- Poll for output: *T* = *T*-1
- If *T* = 0, first message in buffer output

$\mathcal{F}_{\text{A-SFE}}$

Adversary

- Decide on set of *n-t* **input providers**
- Increase *T*, specified in unary

A-SFE Functionality:

- Collects inputs and computes output
- Maintains delay *T*

Modeling Asynchronous BA in UC

Parties P

- Provide input
- Poll for output: T = T-1
- If T = 0, first message in buffer output

$\mathcal{F}_{\text{A-BA}}$

Adversary

- Decide on set C of n-t **input providers**
- Increase T, specified in unary

A-BA Functionality:

- Maintains delay T
- Collects inputs and computes output
  - If there is agreement in C output corresponding value
  - Otherwise, output a value specified by attacker

Our Results

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Protocol Overview

Protocol Overview

- Three phases for computing Boolean circuit C:

Protocol Overview

- Three phases for computing Boolean circuit *C*:
  I. Compute distributed version of garbled circuit
     - Evaluate **constant-depth** function using (unconditionally) secure protocol by [BKR94] (whose round complexity depends on depth of evaluated circuit)

Protocol Overview

- Three phases for computing Boolean circuit *C*:
  I. Compute distributed version of garbled circuit
     - Evaluate **constant-depth** function using (unconditionally) secure protocol by [BKR94] (whose round complexity depends on depth of evaluated circuit)
  II. With output from Phase I, complete circuit garbling

Protocol Overview

- Three phases for computing Boolean circuit C:

  I. Compute distributed version of garbled circuit

  - Evaluate **constant-depth** function using (unconditionally) secure protocol by [BKR94] (whose round complexity depends on depth of evaluated circuit)

  II. With output from Phase I, complete circuit garbling

  III. Locally evaluate garbled circuit

Circuit Garbling [Yao86,BMR90]

- **Idea:** Associated with every wire w of **Boolean** circuit C:
  - mask m_w (to hide actual value on wire) and
  - two keys $k_{w,0}$, $k_{w,1}$
- Evaluate circuit on masked values while maintaining invariant:

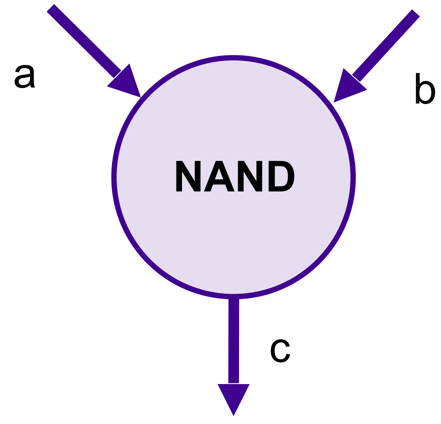
If masked value is z, $k_{w,z}$ is known and $k_{w,1-z}$ is secret

Circuit Garbling [Yao86,BMR90] (2)

z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,0},k_{b,0}, z \,\|\, k_{c,z})$
0	1	$((0 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,0},k_{b,1}, z \,\|\, k_{c,z})$
1	0	$((1 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,1},k_{b,0}, z \,\|\, k_{c,z})$
1	1	$((1 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,1},k_{b,1}, z \,\|\, k_{c,z})$

To evaluate garbled circuit, use:

- Masked values on input wires and corresponding keys
- Masks of output wires



Issue 1

- Evaluating encryption function in MPC → non-constant depth circuit
- **Solution**: “Distributed encryption” [DI05]

Regular encryption: $E(k,m)$

Distributed encryption:

- Use sub-keys $k_1,\ldots,k_n$ instead of k
- Secret-share m
- Give i^{th} share m_i and k_i to party P_i
- P_i computes $E(k_i,m_i)$ and sends to all

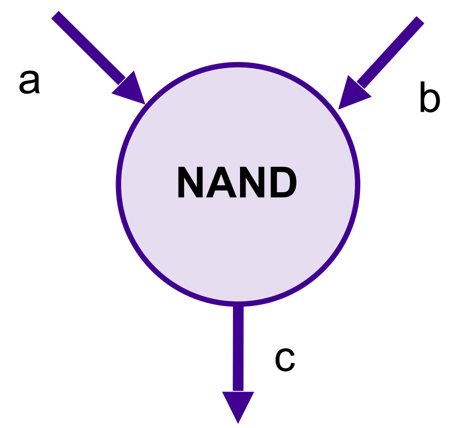
Circuit Garbling with Distributed Encryption

- **Idea:** Associated with every wire w of circuit C:
  - mask m_w (to hide actual value on wire) and
  - two keys $\boldsymbol{k}_{w,0}$, $\boldsymbol{k}_{w,1}$, **each consisting of *n* subkeys**
- Evaluate circuit on masked values while maintaining invariant:

If masked value is z, $\boldsymbol{k}_{w,z}$ is known and $\boldsymbol{k}_{w,1-z}$ is secret.

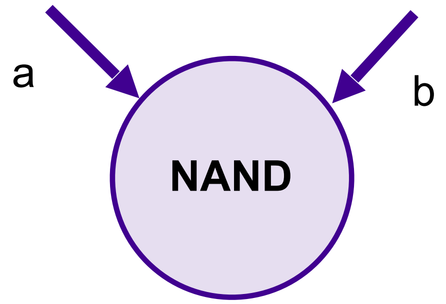
Circuit Garbling without Distributed Encryption

z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,0}, k_{b,0}, z \,\|\, k_{c,z})$
0	1	$((0 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,0}, k_{b,1}, z \,\|\, k_{c,z})$
1	0	$((1 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,1}, k_{b,0}, z \,\|\, k_{c,z})$
1	1	$((1 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,1}, k_{b,1}, z \,\|\, k_{c,z})$



Circuit Garbling with Distributed Encryption

z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$[\, z \,,\, \boldsymbol{k}_{c,z} \,]$
0	1	$((0 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$[\, z \,,\, \boldsymbol{k}_{c,z} \,]$
1	0	$((1 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$[\, z \,,\, \boldsymbol{k}_{c,z} \,]$
1	1	$((1 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$[\, z \,,\, \boldsymbol{k}_{c,z} \,]$



Instead of encrypting garbled entry, compute secret-sharing of (each component of) it

Phase I: Garbling with Distributed Encryption

Phase I: described by (randomized) constant-depth function that

- Randomly chooses masks and subkeys
- Computes masked inputs and corresponding subkeys based on player inputs and masks
- Computes shared function tables (can be done in parallel)
- Outputs to P_i:
  - Masked inputs and corresponding subkeys
  - i^{th} shares of all shared function tables
  - Masks of output wires

Phase I: Garbling with Distributed Encryption

- Actual Phase I: Evaluate Phase I function using [BKR94] protocol
- Round complexity of [BKR94] depends on evaluated circuit
- But: Phase I function is constant-depth

Phases II + III: Encrypting and Evaluating

- Phase II: Compute threshold encryption of garbled entries
  - Each party P_i locally encrypts its shares with the appropriate subkeys and sends resulting ciphertexts to all
- Phase III: Locally evaluate garbled circuit
  - Decryption of a function table entry with decryption subkeys $k_1,\ldots,k_n$:
    - Upon receiving encrypted share from P_i, decrypt it with k_i
    - Wait until $2t+1$ shares on degree-t polynomial received and interpolate

Issue 2

- [BKR94] protocol evaluates arithmetic circuits
- Phase I function described by Boolean circuit
- $\rightarrow$ Conversion to circuit over extension field of GF(2)
  - Replace each NAND gate with inputs x,y by a computation of $1-xy$
- Ensure that all inputs are 0,1 as follows:
  - After input phase, for every input x, jointly open $x - x^2$ [BGN05]
  - If result is 0, accept x, otherwise replace by 0

References

- [Bra84] Gabriel Bracha. An asynchronous ⌊(n – 1)/3⌋-resilient con- sensus protocol. In Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), pages 154–162, 1984. P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer Science Research*, pages 313–322, 1992. Preliminary version in STOC'89.
- [FLP85] M. Fisher, N. Lynch, M. Paterson. Impossibility of Distributed Consensus with one faulty process. JACM, Vol. 32, No. 2, 1985, pp. 374—382
- [BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proc. 25th ACM Symposium on the Theory of Computing (STOC)*, pages 52–61, 1993. Full version in Ran Canetti's PhD Thesis
- [BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proc. 13th ACM Symposium on Principles of Dis- tributed Computing (PODC)*, pages 183–192, 1994.
- [HNP05] M. Hirt, J. Buus Nielsen, and B. Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer-Verlag, May 2005.
- [BH07] Z. Beerliova´-Trub´ıniova´ and M. Hirt. Simple and efficient perfectly-secure asynchronous MPC. In Kaoru Kuro- sawa, editor, *Advances in Cryptology — ASIACRYPT 2007*, vol- ume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer-Verlag, December 2007.

References

- [HNP08] M. Hirt, J. Buus Nielsen, and B. Przydatek. Asynchronous multi-party computation with quadratic com- munication. In Luca Aceto, Magnus M. Halldorsson, and Anna Ingolfsdottir, editors, *Automata, Languages and Program- ming — ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485. Springer-Verlag, July 2008.
- [CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. 2013. Asynchronous Multiparty Computation with Linear Communication Complexity. In *Proceedings of the 27th International Symposium on Distributed Computing - Volume 8205* (DISC 2013), Yehuda Afek (Ed.), Vol. 8205. Springer-Verlag New York, Inc., New York, NY, USA, 388-402. DOI=http://dx.doi.org/10.1007/978-3-642-41527-2_27
- [CB15] Ashish Choudhury and Arpita Patra. 2015. Optimally Resilient Asynchronous MPC with Linear Communication Complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking* (ICDCN '15). ACM, New York, NY, USA, Article 5, 10 pages. DOI: https://doi.org/10.1145/2684464.2684470
- [Coh16] R. Cohen. Asynchronous secure multiparty computation in constant time. In: Public-Key Cryptography - PKC 2016, Proceedings, Part II. pp. 183–207, 2016.
- [CGHZ16] S. Coretti, J. A. Garay, M. Hirt, and V. Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In J. H. Cheon and T. Takagi, editors, ASIACRYPT 2016, volume 10032 of LNCS, pages 998–1021, 2016.