CS4501 Cryptographic Protocols
Lecture 22: DS Lower Bound
Randomized Broadcast

https://jackdoerner.net/teaching/#2026/Spring/CS4501

Roadmap for the Last Few Lectures

Lecture 20, Theorem 1: When $t \geq n/3$, there does not exist a t-secure BA protocol in the plain model, even assuming authenticated channels.

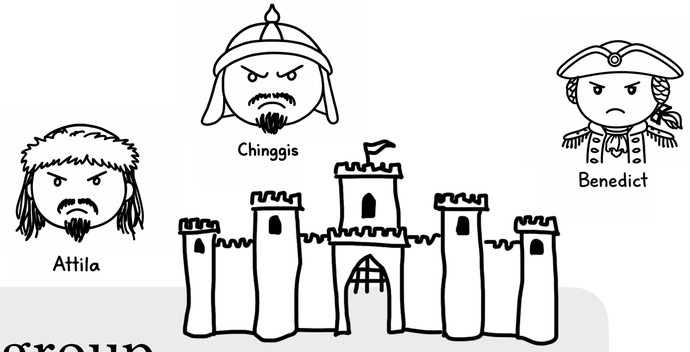
Lecture 20, Theorem 2: When $t < n/3$, there exists a perfectly t-secure BA protocol in the plain model, assuming authenticated channels.

Lecture 21, Theorem 2: Assuming the existence of one-way functions and a *public-key infrastructure*, there exists a (computationally) t-secure broadcast protocol in the setting where $t < n$.

Lecture 22, Theorem 2: Any t-secure *deterministic* broadcast protocol requires at least $t + 1$ rounds.

Lecture 22, Theorem 3: Assuming *trusted* PKI and *verifiable random functions*, there is a *randomized* BA protocol with expected-constant rounds.

Recall: BG/Broadcast



Definition 1. Let $n, t \in \mathbb{N}$ such that $t < n$, and let $P_1, \ldots, P_n$ be a group of parties that are connected by synchronous, authenticated channels. Let π be a protocol in which P_1 has an input $x \in \{0,1\}$ and every P_i produces an output $y_i \in \{0,1\}$. We say that π is a t-secure *Broadcast* protocol if the following conditions hold with all-but-negligible probability when $\mathcal{A}$ maliciously corrupts up to t parties:

1. **Consistency:** All honest parties output the same bit y.

2. **Validity:** If P_1 is honest, then $y = x$.

A broadcast *functionality* simply takes an input from one party and outputs it to all others. In other words, a simulation-secure broadcast protocol is one that securely computes $f(m, \lambda, \ldots, \lambda) = (m, m, \ldots, m)$.

The Dolev-Strong Broadcast Protocol

What's next? We will answer three questions:

1. How can we replace the ideal signatures (which imply an unrealistic restriction on the adversary) with standard EUF-CMA signatures?

2. How can we realize the broadcast functionality (so that we can use our composition theorem to build other protocols on top of broadcast), instead of just achieving a property-based notion of broadcast?

3. Do we really need so many rounds?

2. Realizing the Broadcast Functionality

Recall that our functionality $\mathcal{F}_{\mathsf{BC}}$ simply takes an input from P_1 and sends it to $P_1, \ldots, P_n$. It turns out that *any* protocol that satisfies the properties of broadcast also realizes this functionality in the presence of a malicious adversary.

For every $\mathcal{A}$, we must construct a simulator $\mathcal{S}$ such that the ideal experiment is indistinguishable from the real one. Usually $\mathcal{S}$ emulates an instance of the real experiment toward a copy of $\mathcal{A}$ that $\mathcal{S}$ runs in its head as a subroutine.

The job of $\mathcal{S}$ is to *extract* an input (and output) from each of the emulated parties that $\mathcal{A}$ corrupts, and send that input and output to the corresponding corrupt party in the ideal world, while also producing messages on behalf of the honest parties in the emulated world in such a way that the inputs and outputs of *all* parties in the ideal world are consistent with the transcript observed by the emulated $\mathcal{A}$.

The *easiest* way to emulate an instance of the real experiment toward $\mathcal{A}$ would be to run all of the honest parties using their honest code, but that doesn't usually work…

2. Realizing the Broadcast Functionality

Lemma 1: if π_{BC} is a t-secure broadcast protocol in the presence of some class of adversaries, then it realizes $\mathcal{F}_{\mathsf{BC}}$ in the presence of that same class.

Proof Sketch: $\mathcal{S}$ emulates a copy of the real world experiment involving $\mathcal{A}$, and corrupts the parties in the ideal world that correspond to the ones $\mathcal{A}$ corrupts, as usual, and forwards communication between $\mathcal{A}$ and $\mathcal{D}$. There are two cases:

- P_1 is honest. In this case, $\mathcal{S}$ receives some (consistent) output x from on $\mathcal{F}_{\mathsf{BC}}$ behalf of all the corrupt parties in the ideal world (it doesn't supply any input, because $\mathcal{F}_{\mathsf{BC}}$ only accepts input from P_1).

  $\mathcal{S}$ emulates the honest parties in the real world experiment using their *honest code* and the input x, and the *validity* of π_{BC} ensures that the outputs in the emulated real experiment are consistent with those in the ideal experiment, regardless of the behavior of $\mathcal{A}$.

  Thus the information received by $\mathcal{D}$ is distributed identically to an instance of the real-world experiment where honest P_1 has input x.

2. Realizing the Broadcast Functionality

- P_1 is honest. In this case, $\mathcal{S}$ receives some (consistent) output x from on $\mathcal{F}_{\mathsf{BC}}$ behalf of all the corrupt parties in the ideal world (it doesn't supply any input, because $\mathcal{F}_{\mathsf{BC}}$ only accepts input from P_1).

  $\mathcal{S}$ emulates the honest parties in the real world experiment using their *honest code* and the input x, and the *validity* of π_{BC} ensures that the outputs in the emulated real experiment are consistent with those in the ideal experiment, regardless of the behavior of $\mathcal{A}$.

  Thus the information received by $\mathcal{D}$ is distributed identically to an instance of the real-world experiment where honest P_1 has input x.

- P_1 is corrupted. In this case, $\mathcal{S}$ must *supply* input x to $\mathcal{F}_{\mathsf{BC}}$ before it will run. $\mathcal{S}$ emulates the honest parties in the real world experiment using their honest code, and the *consistency* of π_{BC} ensures that all emulated honest parties output a single, consistent value x at the end of the protocol, regardless of the behavior of $\mathcal{A}$.

  $\mathcal{S}$ sends x to $\mathcal{F}_{\mathsf{BC}}$ on behalf of the corrupted P_1, causing the honest parties to output it in the ideal world as well. Thus the information received by $\mathcal{D}$ is distributed identically to an instance of the real-world experiment where P_1 is corrupted. ∎

2. Realizing the Broadcast Functionality

Lemma 1: if π_{BC} is a t-secure broadcast protocol in the presence of some class of adversaries, then it realizes $\mathcal{F}_{\mathsf{BC}}$ in the presence of that same class.

Recall **Lecture 21 Theorem 2:** Let $n, t \in \mathbb{N}$ such that $t < n$. Assuming ideal digital signatures and a public-key infrastructure, there exists a t-secure broadcast protocol in the presence of any PPT adversary.

Corollary 1: Let $n, t \in \mathbb{N}$ such that $t < n$. Assuming ideal digital signatures and a public-key infrastructure, there exists a protocol that realizes $\mathcal{F}_{\mathsf{BC}}$ in the presence of any PPT adversary that corrupts up to t parties.

Proof: By Lemma 1 and Lecture 21 Theorem 2. ∎

Note: we haven't talked about how $\mathcal{S}$ can deal with *adaptive* corruptions: Notice that since it's just emulating the real-world protocol internally, it can always send the internal state of any party to $\mathcal{A}$ at any point that $\mathcal{A}$ decides to corrupt someone!

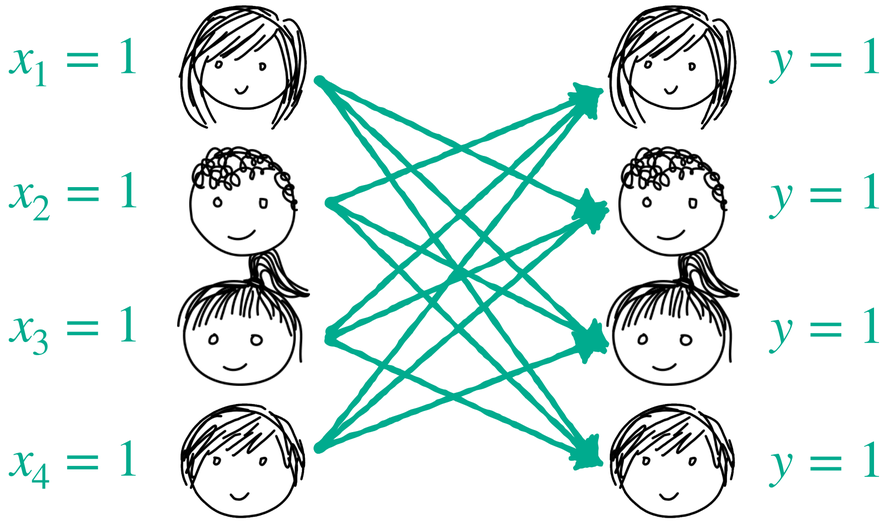
3. How many rounds do we need? (Warm Up)

We will show that if a *deterministic* broadcast protocol has consistency and it terminates in t rounds or fewer, then it *must* always output a constant value, regardless of what the input is. In other words, it cannot have validity.

Before we see the general proof, we'll do a warm-up version.

Theorem 1: There is no 1-round, 4-party, 1-secure deterministic BA protocol.

Proof: Suppose such a protocol existed. Here is a diagram of its messages:

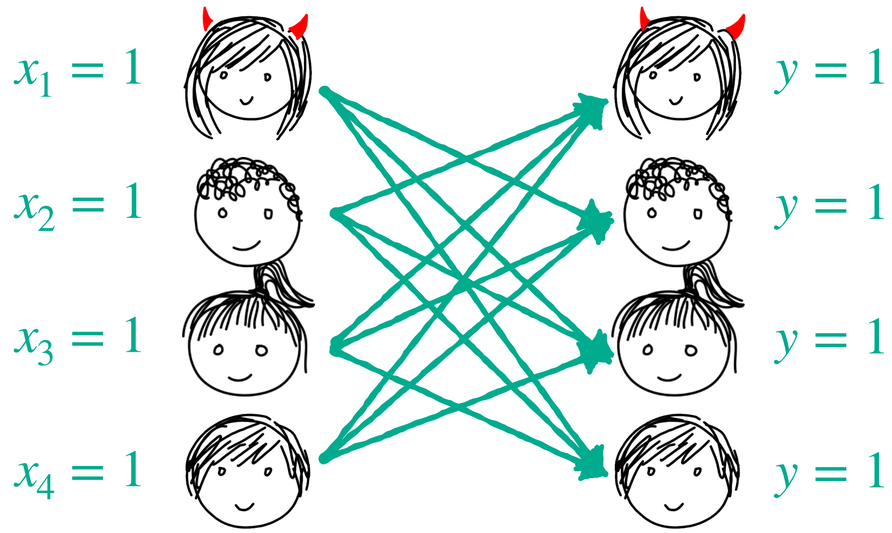


Consider an instance E_1 of the experiment in which everyone has input 1. By validity, all parties must also output 1.

3. How many rounds do we need? (Warm Up)

Theorem 1: There is no 1-round, 4-party, 1-secure deterministic BA protocol.

Proof: Suppose such a protocol existed. Here is a diagram of its messages:

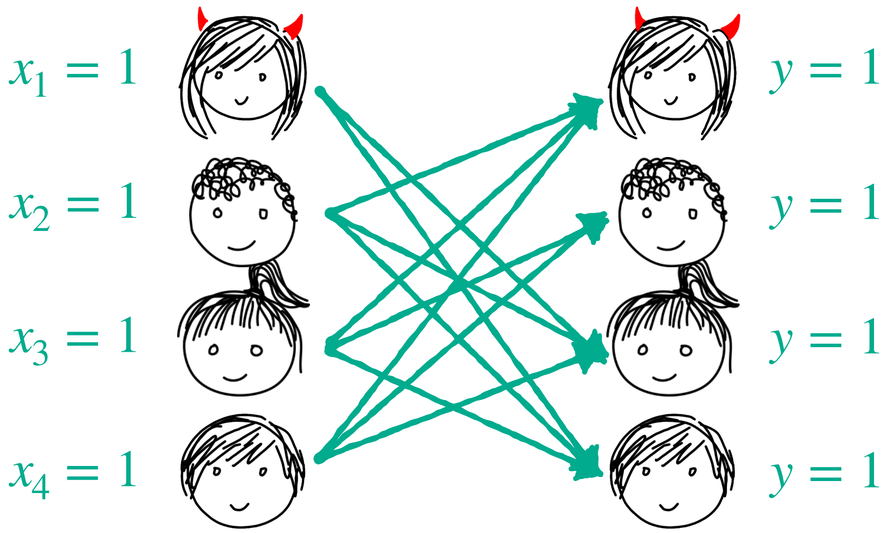


Consider an instance E_1 of the experiment in which everyone has input 1. By validity, all parties must also output 1.

Next, consider instance E_2 in which P_1 is corrupt and fails to send a message to P_2 (but otherwise behaves honestly).

Since the views of P_3 and P_4 have not changed, they must still output 1, and by consistency, P_2 must also still output 1.

3. How many rounds do we need? (Warm Up)



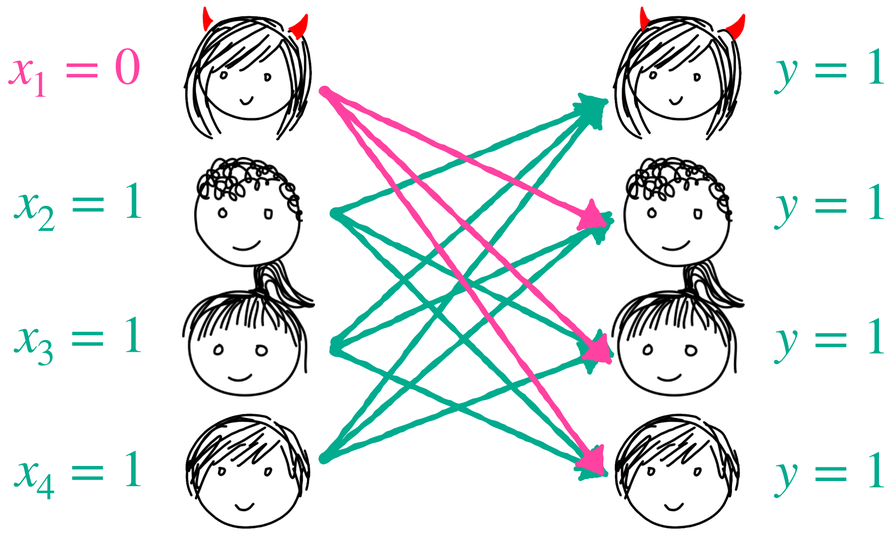
Next, consider instance E_2 in which P_1 is corrupt and fails to send a message to P_2 (but otherwise behaves honestly).

Since the views of P_3 and P_4 have not changed, they must still output 1, and by consistency, P_2 must also still output 1.

Similarly, if E_3 is an instance in which P_1 fails to send to P_3 (in addition to P_2), then relative to E_2, only the view of P_3 has changed, and everyone must still output 1.

Finally, in E_4 we remove the communication from P_1 to P_4. The outputs remain 1.

3. How many rounds do we need? (Warm Up)



Finally, in E_4 we remove the communication from P_1 to P_4. The outputs remain 1.

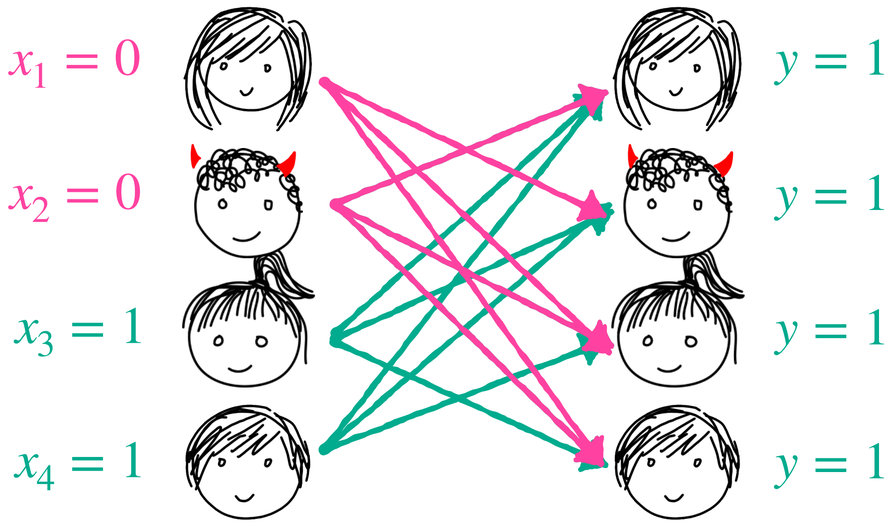
Notice now that P_1 does not influence anyone!

Therefore, we can change her input to $x_1 = 0$ in E_5. The other parties still output 1, because their views do not change, and she still outputs 1, by consistency.

Now, doing the process in reverse, we define E_6, E_7, E_8. In each she sends an honest message (with $x_1 = 0$) to one additional honest party, and none of the outputs change relative to the previous experiment.

In E_8, P_1 is behaving fully honestly!

3. How many rounds do we need? (Warm Up)

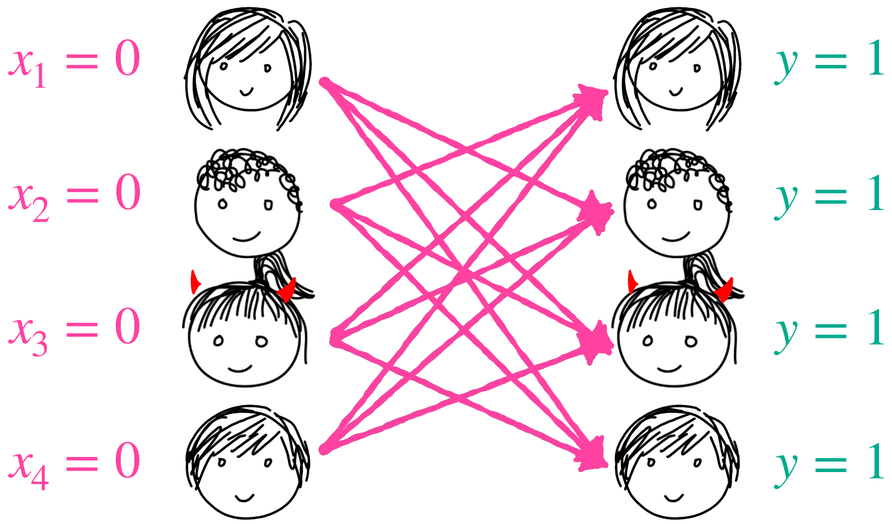


In E_8, P_1 is behaving fully honestly!

We can repeat exactly the same sequence of steps for P_2 in order to replace its input with $x_2 = 0$ while arguing that all parties must output 1.

This defines $E_9, \ldots, E_{15}$.

3. How many rounds do we need? (Warm Up)



In E_8, P_1 is behaving fully honestly!

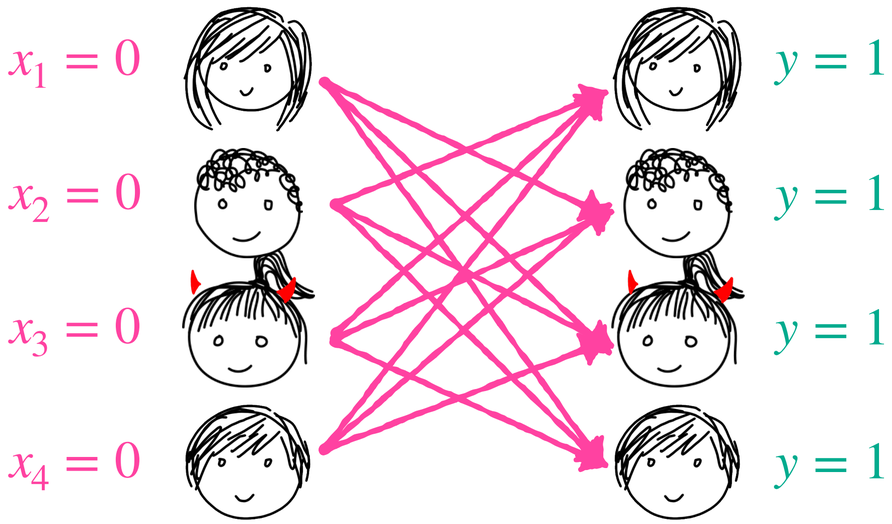
We can repeat exactly the same sequence of steps for P_2 in order to replace its input with $x_2 = 0$ while arguing that all parties must output 1.

This defines $E_9, \ldots, E_{15}$.

Then we can do it for P_3 and then P_4, defining $E_{16}, \ldots, E_{29}$. The same sequence of arguments tells us that in E_{29}, all of the parties must output 1.

But in E_{29}, all of the parties behave fully honestly and have input 0, so this violates the validity of the protocol!

3. How many rounds do we need? (Warm Up)



Then we can do it for P_3 and then P_4, defining $E_{16}, \ldots, E_{29}$. The same sequence of arguments tells us that in E_{29}, all of the parties must output 1.

But in E_{29}, all of the parties behave fully honestly and have input 0, so this violates the validity of the protocol! ∎

Notice that in this proof, the adversary never sends a malformed message. Its only attack is to *fail* to send a message.

Therefore, this proof holds even for *fail-stop* adversaries that either behave honestly or crash. These are weaker than malicious adversaries.

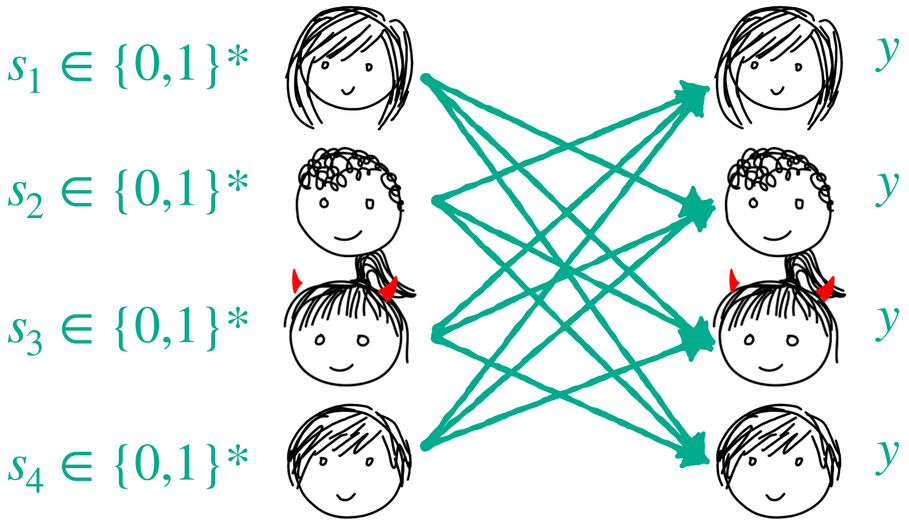
3. How many rounds do we need?

Now we will start generalizing the ideas in the last proof towards capturing an a *arbitrary* t-secure, t -round deterministic broadcast protocol. This will be the implicit context for the next six lemmas.

Notice that nothing in the previous proof requires the parties to start with the same input, or their inputs and outputs to be bits.

We can choose their inputs to be arbitrary non-identical bit strings, and we will arrive at the conclusion that all possible inputs yield the same output bitstring, so long as the protocol is consistent.

Next, we will abstract three ideas from the proof into lemmas about this generalized agreement protocol. We will call the one round it contains round t.



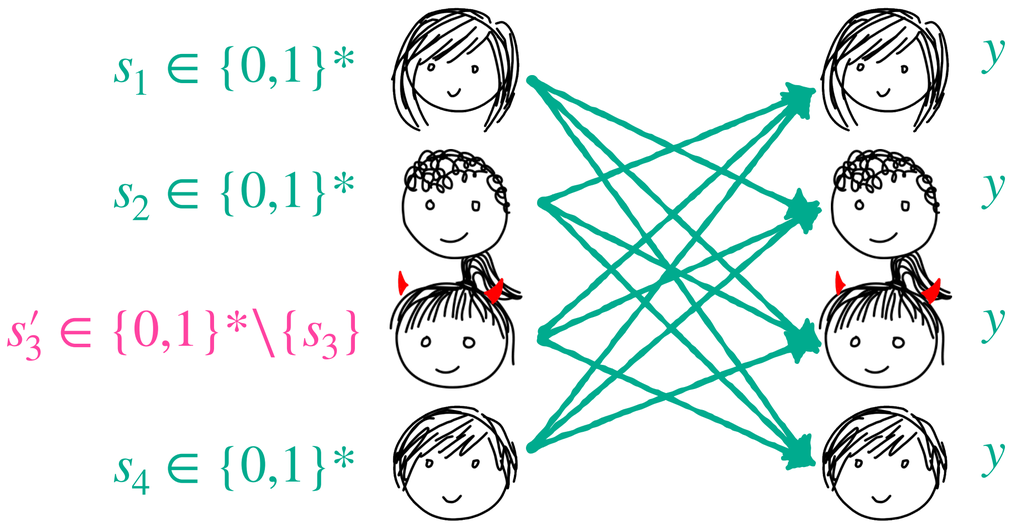
3. How many rounds do we need?

Lemma 2: Suppose we have an experiment E in which a party P_c is corrupt in round t, but behaves honestly.

If we formulate a similar experiment E' in which P_c does not send any messages in round t (or any later round), then all parties must output the same thing in experiments E and E'.

Lemma 3: Suppose we have an experiment E in which a party P_c is corrupt in round t, and starts with state s_c, but is silent from round t onward.

If we formulate a similar experiment E' in which P_c begins round t with a *different* state s'_c, and remains silent from then onward, then all parties must output the same thing in experiments E and E'.



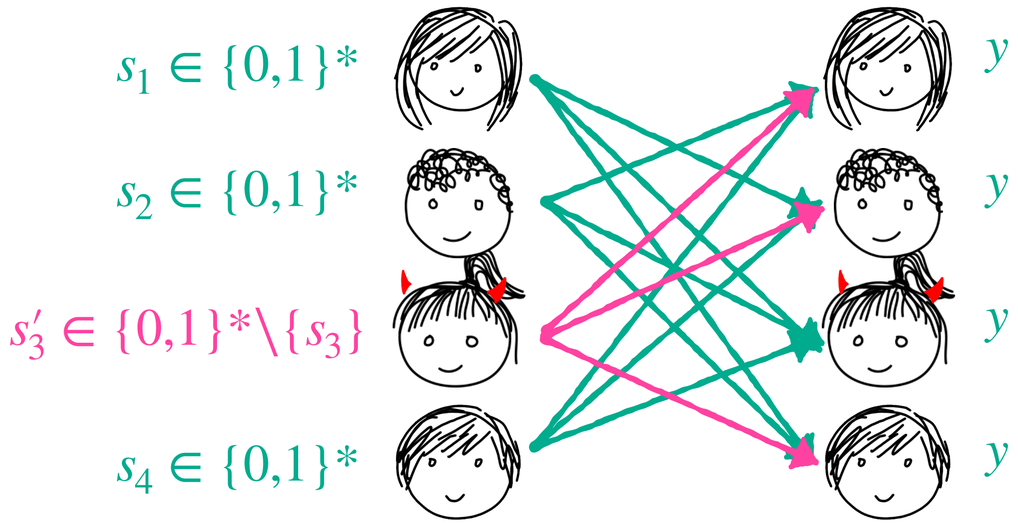
3. How many rounds do we need?

Lemma 3: Suppose we have an experiment E in which a party P_c is corrupt in round t, and starts with state s_c, but is silent from round t onward.

If we formulate a similar experiment E' in which P_c begins round t with a *different* state s'_c, and remains silent from then onward, then all parties must output the same thing in experiments E and E'.

Lemma 4: Suppose we have an experiment E in which a party P_c is corrupt in round t, and starts with an *arbitrary* state s'_c, but is silent from round t onward.

If we formulate a similar experiment E' in which P_c sends the messages in round t that an *honest* party would send given s'_c, then all parties must output the same thing in experiments E and E'.



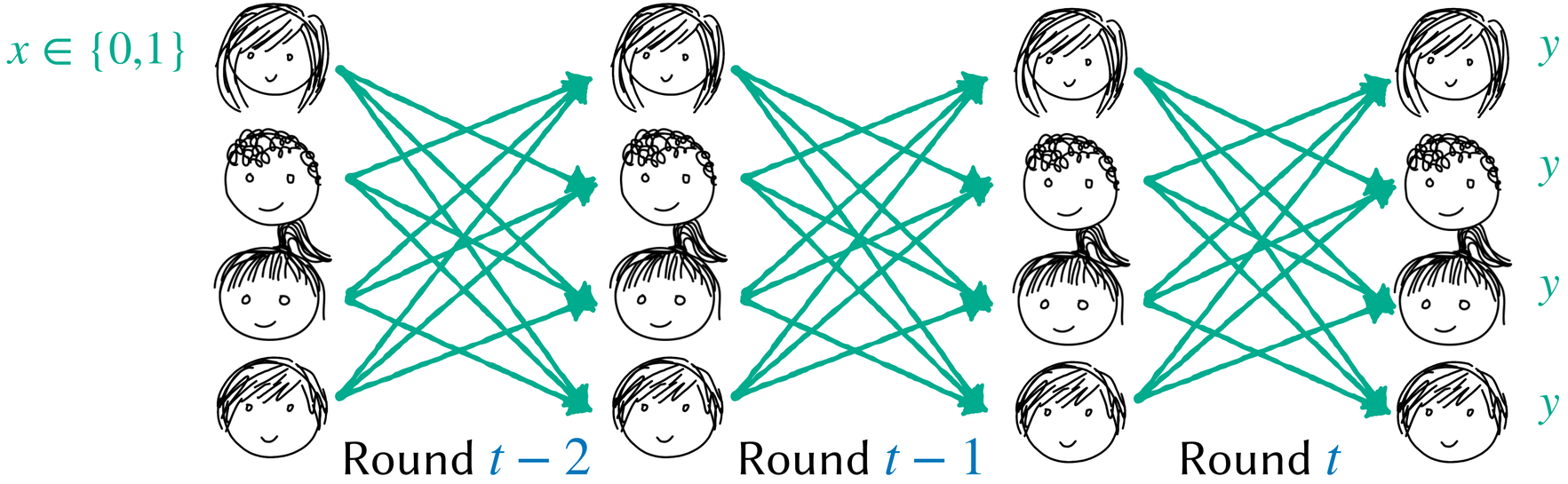
3. How many rounds do we need?

Now think of round t as being the final round of an *arbitrary* t-round t-secure broadcast protocol with a similar communication pattern in all previous rounds.

In each round, a party's honest message is a deterministic function of its state, which is a combination of the messages it received in the previous round, and its previous state. At the beginning of round 1, P_1's state is its input, and all other parties have a blank state.

Next, we will prove that if the three lemmas all hold with respect to some round $r \leq t$, then they all *also* hold with respect to round $r - 1$. We will prove this *by example* using round $t - 1$.

Note: from now on we will define new experiment *implicitly* whenever we "change something."

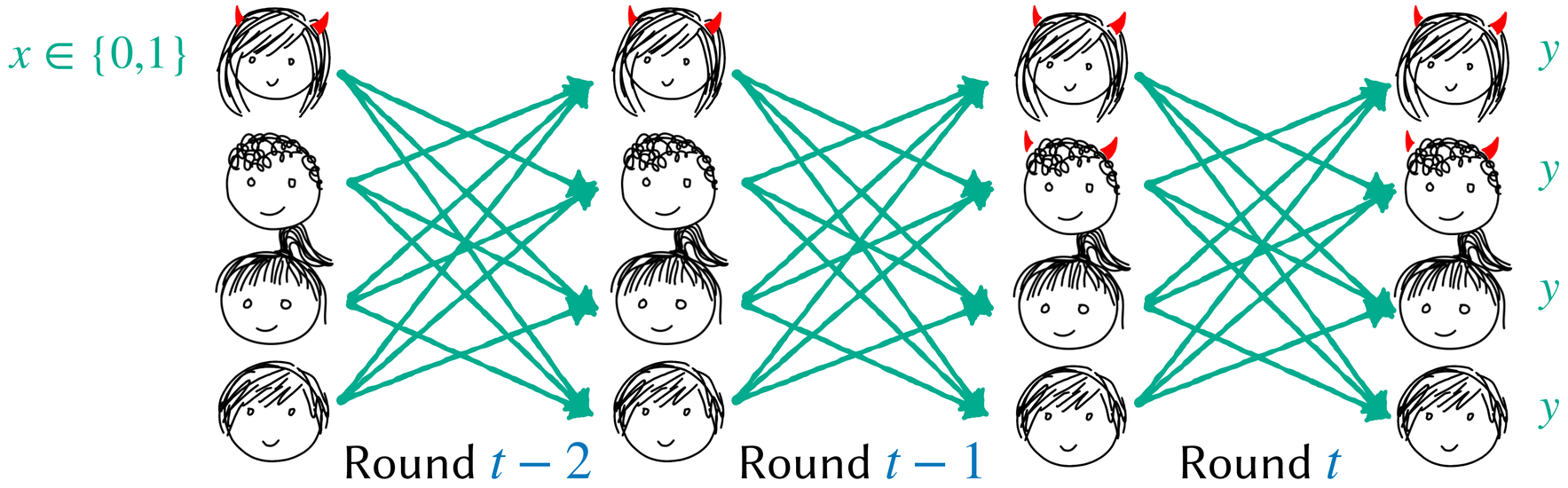


3. How many rounds do we need?

Lemma 5: Suppose we have an experiment E in which a party P_c is corrupt in round $t-1$, but behaves honestly. If we formulate a similar experiment E' in which P_c does not send any messages in round $t-1$ or afterward, then all parties must output the same thing in experiments E and E'.

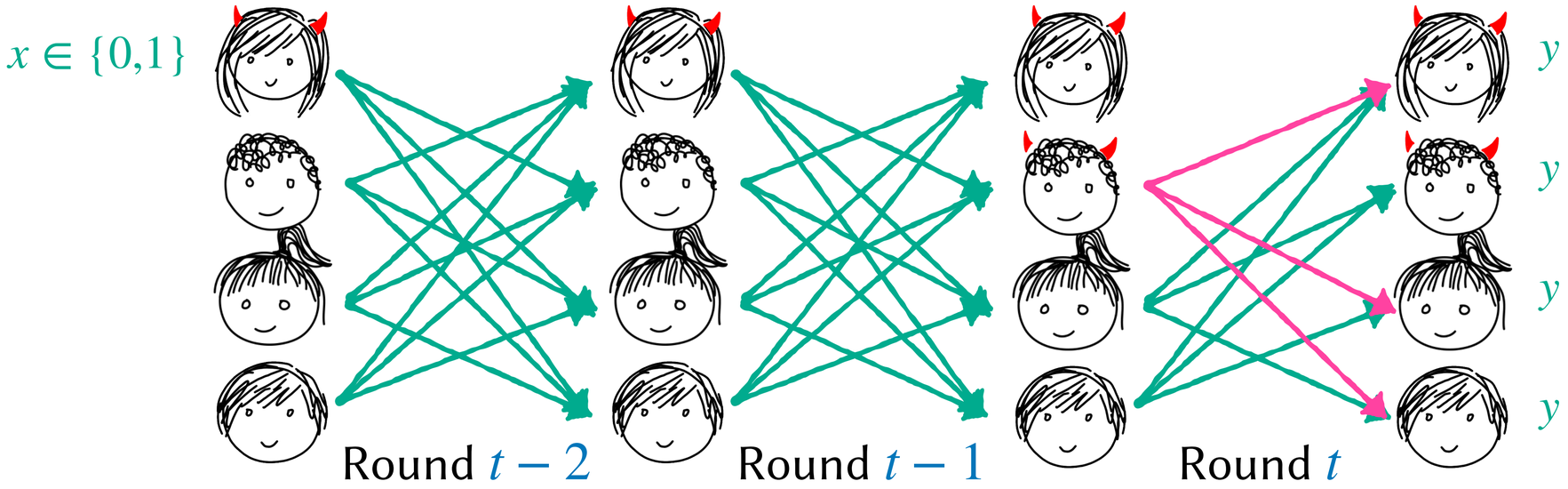
Proof: By Lemma 2, we can eliminate P_c's messages in round t without changing the output.

- The protocol tolerates t corruptions, and we know $t \geq 2$, but we have only considered 1 corruption so far. If we imagine P_{c+1} is *also* corrupt in round t, then we can also use Lemma 2 to eliminate the messages of P_{c+1} in round t, without changing the output.



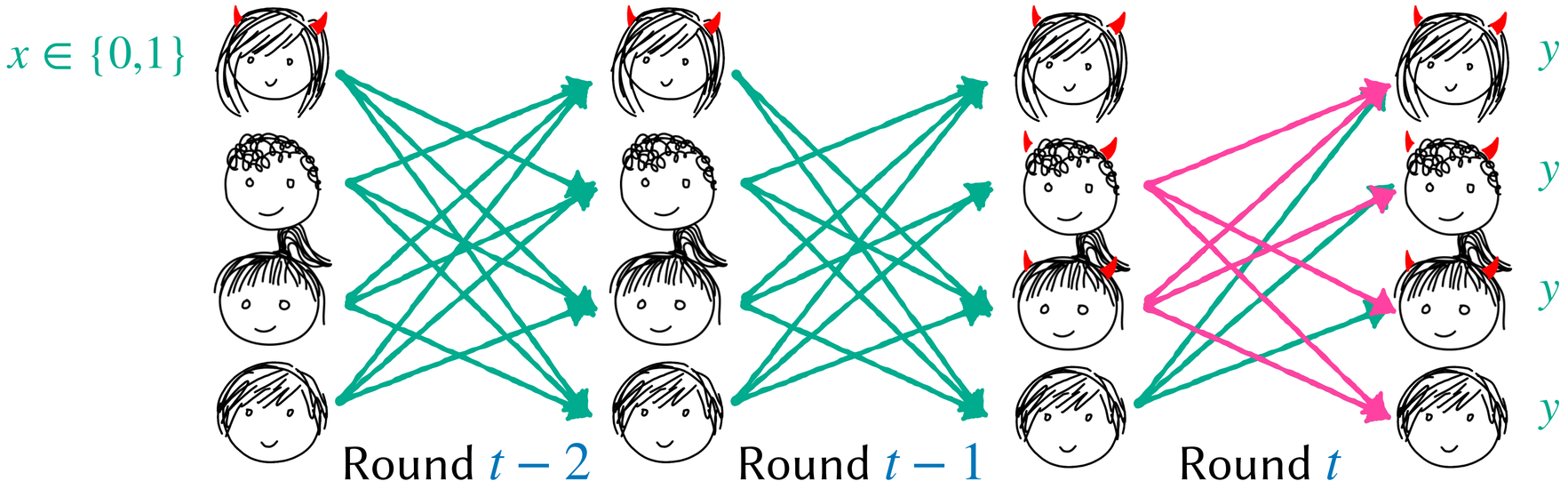
3. How many rounds do we need?

- The protocol tolerates t corruptions, and we know $t \geq 2$, but we have only considered 1 corruption so far. If we imagine P_{c+1} is *also* corrupt in round t, then we can also use Lemma 2 to eliminate the messages of P_{c+1} in round t, without changing the output.
- Next, we eliminate the message from P_c to P_{c+1} in round $t-1$. This changes the *state* of P_{c+1} at the beginning of round t, but by Lemma 3, it *doesn't* impact the protocol output.
- By Lemma 4, we find that if P_{c+1} sends the messages an honest party would send when it doesn't get a message from P_c in round $t-1$, the overall output still doesn't change.



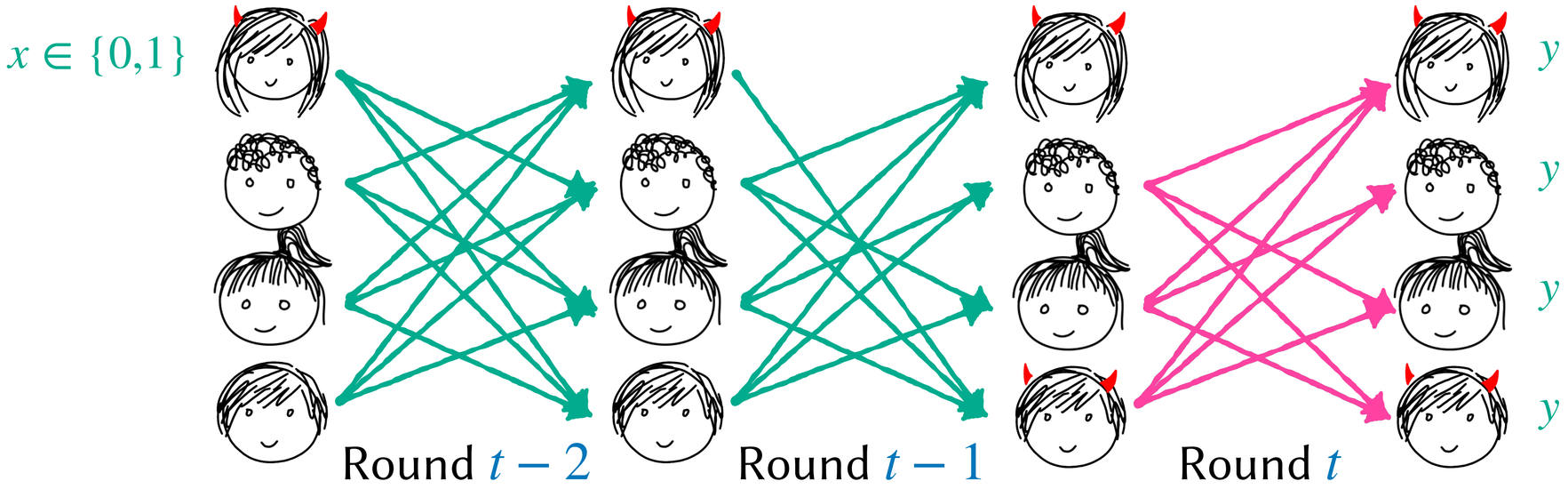
3. How many rounds do we need?

- By Lemma 4, we find that if P_{c+1} sends the messages an honest party would send when it doesn't get a message from P_c in round $t-1$, the overall output still doesn't change.
- Since P_{c+1} now behaves completely honestly in all rounds, the output is the same as if P_{c+1} actually is honest in all rounds.
- Next, we can consider P_{c+1} to be corrupt in round $t-1$, and repeat the above steps to remove the message from P_c to P_{c+1} in round $t-1$, still without changing the output of the protocol.



3. How many rounds do we need?

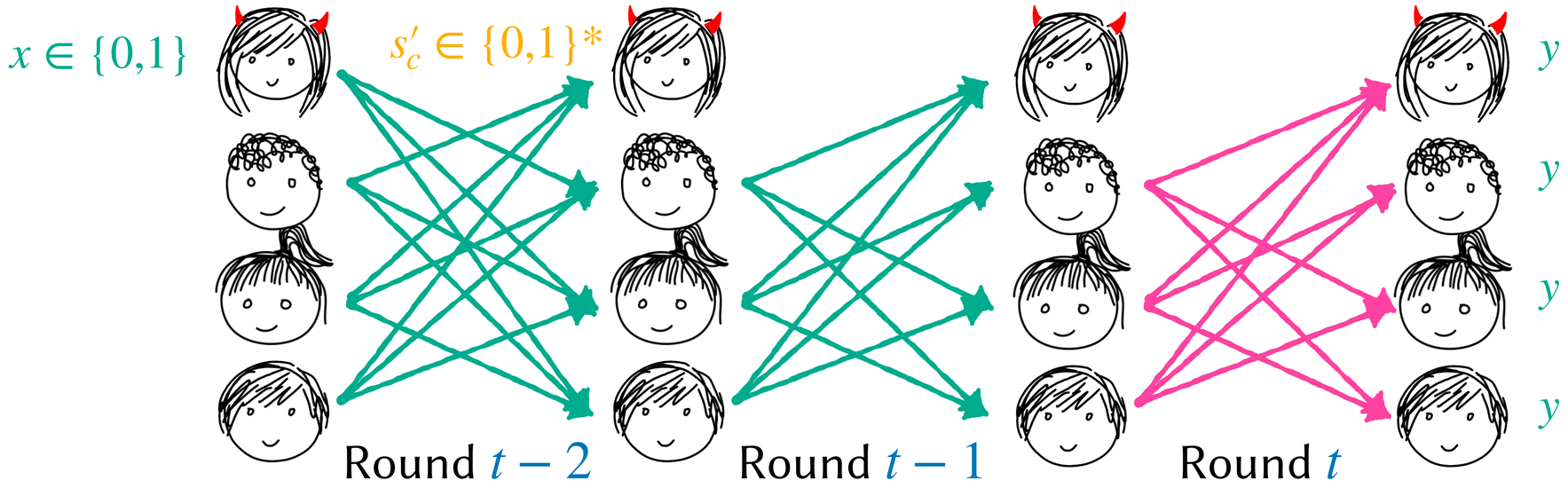
- By Lemma 4, we find that if P_{c+1} sends the messages an honest party would send when it doesn't get a message from P_c in round $t-1$, the overall output still doesn't change.
- Since P_{c+1} now behaves completely honestly in all rounds, the output is the same as if P_{c+1} actually is honest in all rounds.
- Next, we can consider P_{c+1} to be corrupt in round $t-1$, and repeat the above steps to remove the message from P_c to P_{c+1} in round $t-1$, still without changing the output of the protocol.
- We can repeat this until all outbound messages from P_c in round $t-1$ are removed. ∎



3. How many rounds do we need?

Lemma 6: Suppose we have an experiment E in which a party P_c is corrupt in round $t-1$, and is silent from then onward. We can define experiment E' by making an arbitrary change to the state of P_c at the beginning of round $t-1$, and the output of the protocol will be the same in E and E'.

Proof: This follows from exactly the same reasoning as Lemma 3: since P_c has no outbound messages after the state change, the state change cannot affect the output of any other party, and consistency ensures that the output of P_c remains the same in E' as well. ∎

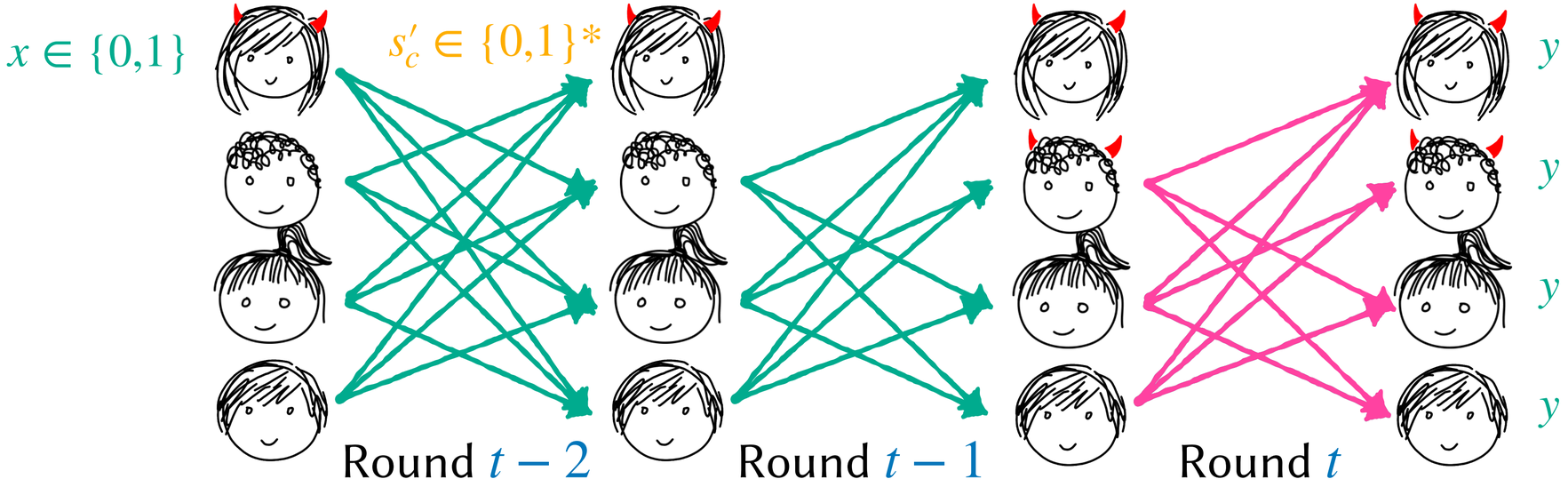


3. How many rounds do we need?

Lemma 7: Suppose we have an experiment E in which a party P_c is corrupt in round $t-1$, and sets its state to an arbitrary s'_c at that point, but is silent from then onward. If we formulate a similar experiment E' in which P_c sends the messages in round $t-1$ and onwards that an *honest* party would send given s'_c, then all parties must output the same thing in experiments E and E'.

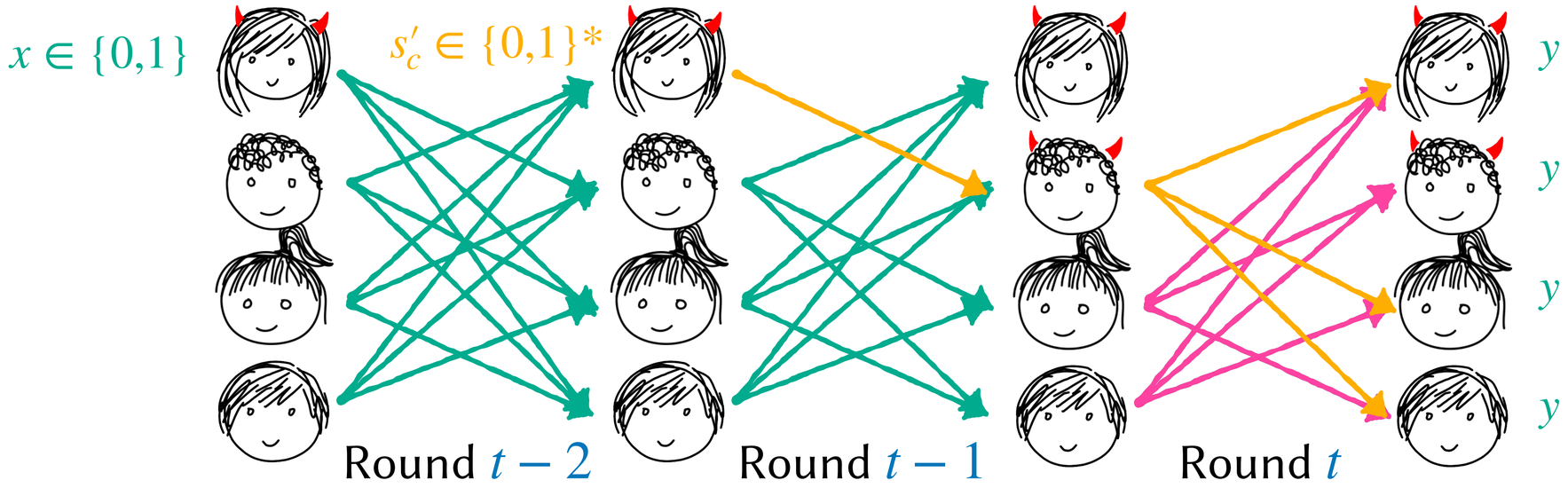
Proof: Our proof is similar to that of Lemma 5. For each P_d such that $d \neq c$, we:

- First imagine that P_d is corrupt but behaving honestly (and therefore no change in output).
- Then use Lemma 2 to eliminate the messages of P_d in round t, without changing the output.



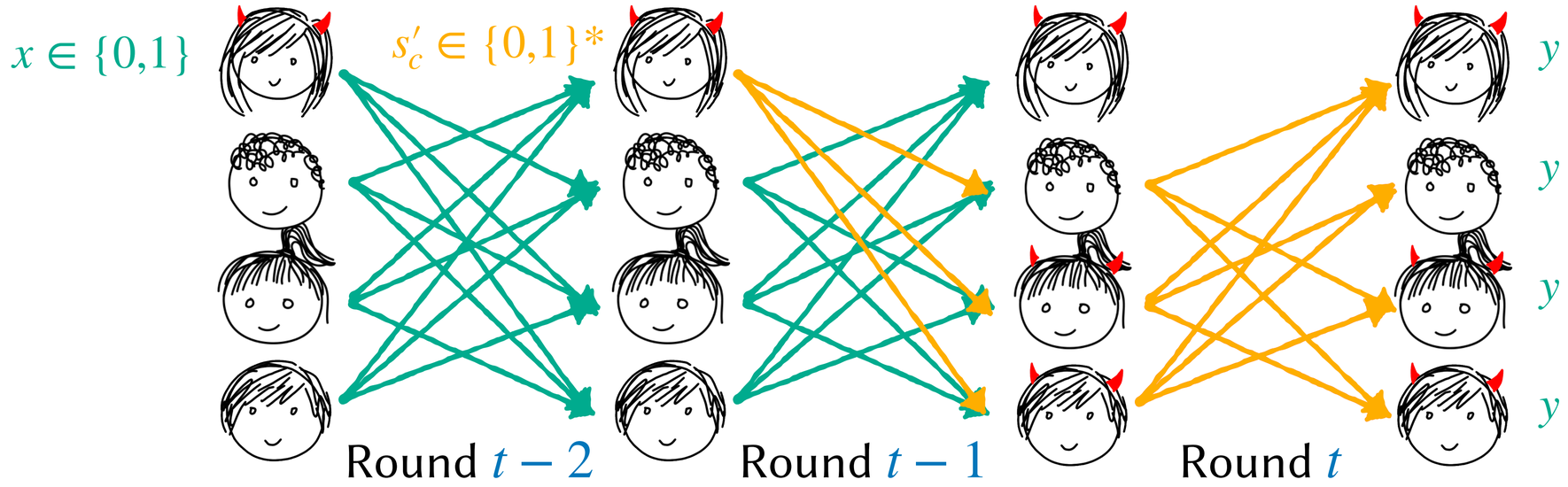
3. How many rounds do we need?

- Then use Lemma 2 to eliminate the messages of P_d in round t, without changing the output.
- Then add the honestly-computed message from P_c to P_d in round $t-1$. By Lemma 3, this has no effect on the output of the protocol, though it changes the state of P_d in round t.
- Use Lemma 4 to add back the honestly-computed messages of P_d in round t.
- Finally, since P_d behaves completely honestly, we can conclude that all outputs would be the same if he were honest.



3. How many rounds do we need?

- Then use Lemma 2 to eliminate the messages of P_d in round t, without changing the output.
- Then add the honestly-computed message from P_c to P_d in round $t-1$. By Lemma 3, this has no effect on the output of the protocol, though it changes the state of P_d in round t.
- Use Lemma 4 to add back the honestly-computed messages of P_d in round t.
- Finally, since P_d behaves completely honestly, we can conclude that all outputs would be the same if he were honest.
- Repeating these sets for every P_d such that $d \neq c$, gives us the lemma. ∎

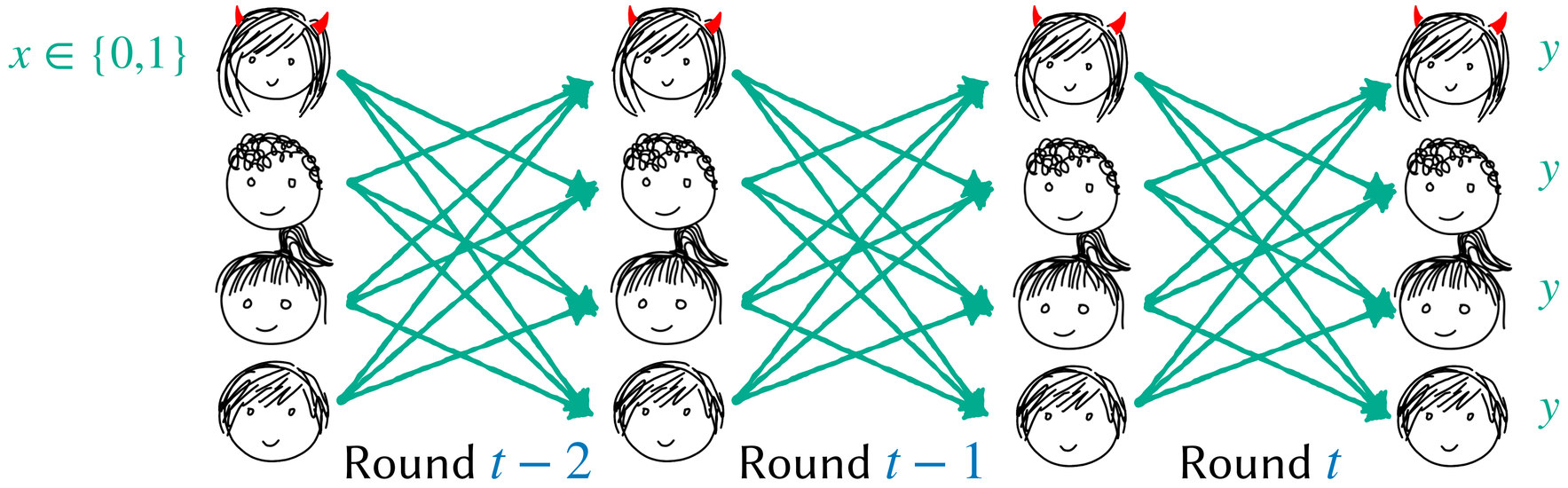


3. How many rounds do we need?

Finally! Theorem 2: There is no t-secure deterministic broadcast protocol with $\leq t$ rounds.

Proof: First, note that we can apply Lemmas 5,7 *inductively* using Lemmas 2,3,4,6 as base cases, and extend the result to an arbitrary number of rounds! To make our induction valid for each additional round (counting from the last one), our protocol must handle one *additional* corruption.

Let us assume that there exists a t-round t-secure deterministic broadcast protocol. We can perform t layers of induction, and remove *all* of the messages of P_1 without changing the output.

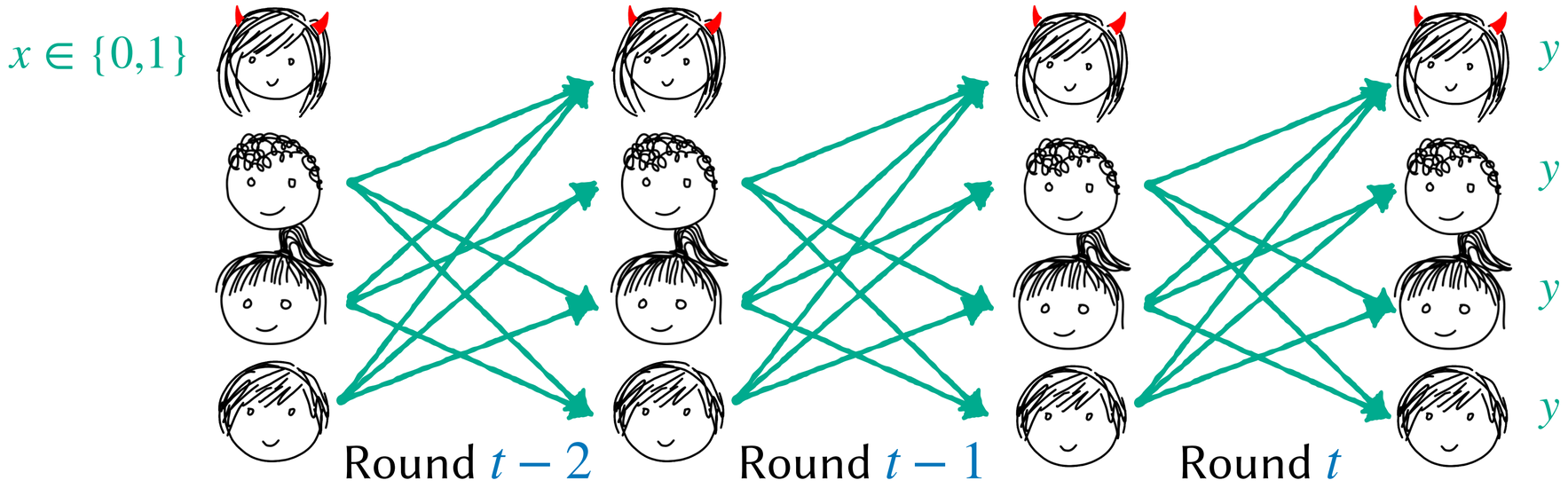


3. How many rounds do we need?

Proof: First, note that we can apply Lemmas 5,7 *inductively* using Lemmas 2,3,4,6 as base cases, and extend the result to an arbitrary number of rounds! To make our induction valid for each additional round (counting from the last one), our protocol must handle one *additional* corruption.

Let us assume that there exists a t-round t-secure deterministic broadcast protocol. We can perform t layers of induction, and remove *all* of the messages of P_1 without changing the output.

Thus, *any* honest execution on *any* input $x \in \{0,1\}$ must produce the same output as the execution in which P_1 never sends any messages. This clearly contradicts validity! ∎



Notice that the previous proof requires that honest parties be *deterministic*. The Dolev-Strong protocol is deterministic (in the ideal signatures model), and therefore optimal.

The BGP protocol we saw before is *also* deterministic and thus asymptotically optimal (with a small coefficient).

These solutions are OK if you and a few frenemies want to go to space or lay siege to a city. *What if we want the whole world to agree on something?* e.g. Bitcoin had ~20 million users in 2025 (per Andreson Horowitz)

Can we do better if we introduce randomness?

Recall: The BGP Protocol for $t < n/3$

Protocol π_1: In this protocol, P_1 is the king.

1. Each P_i sends x_i to everyone. If P_i receives a value b from $\geq n - t$ parties, then it sets $v_i := b$. Otherwise it sets $v_i := \perp$ (undecided).

2. Each P_i sends v_i to everyone if $v_i \neq \perp$. If P_i receives a value b from $\geq n - t$ parties, then it sets $w_i := b$, $g_i = 2$. Otherwise, if P_i receives a value b from $\geq t + 1$ parties, it sets $w_i := b$, $g_i = 1$. Otherwise, P_i sets $w_i := x_i$, $g_i = 0$.

3. The king (P_1) sends w_1 to everyone. If P_i receives b from P_1, and $g_i < 2$, then P_i sets $y_i := b$. Otherwise $y_i := w_i$. Finally, every P_i outputs y_i.

Recall: if the king is honest, they help the other parties us get consistency in step 3. We have to repeat this process $t + 1$ times to be sure we get an honest king.

But 2/3 of parties are honest! *What would happen if we chose our leader at random?*

Recall: The BGP Protocol for $t < n/3$

Recall: if the king is honest, they help the other parties us get consistency in step 3. We have to repeat this process $t + 1$ times to be sure we get an honest king.

But 2/3 of parties are honest! *What would happen if we chose our leader at random?*

Intuition: when we have a *constant fraction* of parties, if we could pick a leader at random, then we could come to agreement in *constant rounds* in expectation.

Question: But agreeing on a leader would seem to imply *agreement!* Are we back where we started?

Not quite! If we fail to agree, the consequence is that we don't achieve consistency and we have to keep going (i.e. the same as if the leader is dishonest). It's sufficient to elect a consistent, honest leader *with constant probability*.

(Also, as a technical note, our bounds on agreement don't necessarily apply to the case where the output is a *random* value, rather than an input of the honest parties!)

We will build a mechanism that *sometimes* elects a (pseudo)random, consistent, honest leader.

Towards Pseudorandom Leader Election

Recall: in Lecture 16 we introduced the notion of *pseudorandom* functions. These are, in short, function families whose members have *compact descriptions*, yet if member of the family is chosen at random, then its outputs are indistinguishable from the outputs of a function that maps all inputs to random values.

Note that PRFs guarantee *nothing* if the adversary gets to choose the function. It is legal for some member of the family to (for example) always output 0, so long as that member is chosen with negligible probability.

Suppose $\mathscr{A}$ is semi-honest, and we have a PRF family $\mathfrak{F} = \left\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\right\}_{k \in \{0,1\}^*}$.
Then we could elect a leader in the following way:

1. At the beginning of time, every party P_i samples $k_i \leftarrow \{0,1\}^\kappa$ uniformly.

2. To elect a leader in Round r, every P_i computes $\gamma_i := F_{k_i}(r)$ and broadcasts γ_i. The leader for round r is P_{i^*} where $i^* = \text{argmin}_{i \in [n]}(\gamma_i)$. That is, you win the election by having the smallest value of γ_i.

Towards Pseudorandom Leader Election

Suppose $\mathscr{A}$ is semi-honest, and we have a PRF family $\mathfrak{F} = \left\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\right\}_{k \in \{0,1\}^*}$.
Then we could elect a leader in the following way:

1. At the beginning of time, every party P_i samples $k_i \leftarrow \{0,1\}^\kappa$ uniformly.

2. To elect a leader in Round r, every P_i computes $\gamma_i := F_{k_i}(r)$ and broadcasts γ_i. The leader for round r is P_{i^*} where $i^* = \text{argmin}_{i \in [n]}(\gamma_i)$. That is, you win the election by having the smallest value of γ_i.

Notice that from the perspective of $\mathscr{A}$, the value of γ_h for any honest party P_h is indistinguishable from random! Furthermore, every party has a (negligibly close to) equal probability of winning the election in every round, over the coins that are used to sample all of the PRF keys.

If $\mathscr{A}$ is malicious we have two major problems that we need to solve:

1. A corrupt party P_c could *lie* about the value of γ_c.

2. A corrupt party P_c can choose its *favorite* k_c instead of a random one, such that F_{k_c} misbehaves.

Towards Pseudorandom Leader Election

If $\mathscr{A}$ is malicious we have two major problems that we need to solve:

1. A corrupt party P_c could *lie* about the value of γ_c.

2. A corrupt party P_c can choose its *favorite* k_c instead of a random one, such that F_{k_c} misbehaves.

Solving Problem 1: here we will take some inspiration from *Digital Signature Schemes*, which were introduced in the last lecture. A signature scheme makes use of both a public key and a secret key. The public key is used to verify that some particular message/signature pair is *valid* for the corresponding secret key, without revealing what the secret key is.

We can ask for a similar property here. We will endow every member of the PRF family with a public key (and rename k to sk), and use the plubic key to verify that $y = F_{\mathsf{sk}}(x)$, given x and y.
In a moment, we will see how to make this fit together with the pseudorandomness property.

We call a function family with these properties a *verifiable random function* (VRF). For now we will simply assume that it can be constructed. In a few more lectures, we will see a technique called *zero-knowledge proofs* that allow us to construct a VRF from any PRF.

Towards Pseudorandom Leader Election

If $\mathscr{A}$ is malicious we have two major problems that we need to solve:

1. A corrupt party P_c could *lie* about the value of γ_c.

2. A corrupt party P_c can choose its *favorite* k_c instead of a random one, such that F_{k_c} misbehaves.

Solving Problem 2: in general, we could imagine solving this by preventing P_c from biasing the choice of k_c too much, or by designing the function family in such a way that *every* member F_{k_c} is well-behaved.

In this class we'll take the former approach, for the sake of simplicity. Specifically, we will *assume* that the uniform sampling of k_c is enforced when the PKI setup phase occurs.

In practice, enforcing this kind of restriction is difficult (especially when broadcast is not available). Instead, we can use VRFs with an extra (strong!) security property that guarantees its outputs cannot be detectably biased, even if the adversary has free choice of k_c.

This special unbiasable VRF was described by Giunta and Stewart in 2024! It's out of scope for us.

Verifiable Random Functions

Def 1 (VRF). A trio of algorithms $\Psi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a *verifiable random function* if:

1. Gen and Prove are PPT, and Verify is deterministic polynomial time.

2. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$ samples a key pair such that $\mathsf{Prove}_{\mathsf{sk}} : \{0,1\}^\kappa \to \{0,1\}^\kappa \times \{0,1\}^*$.

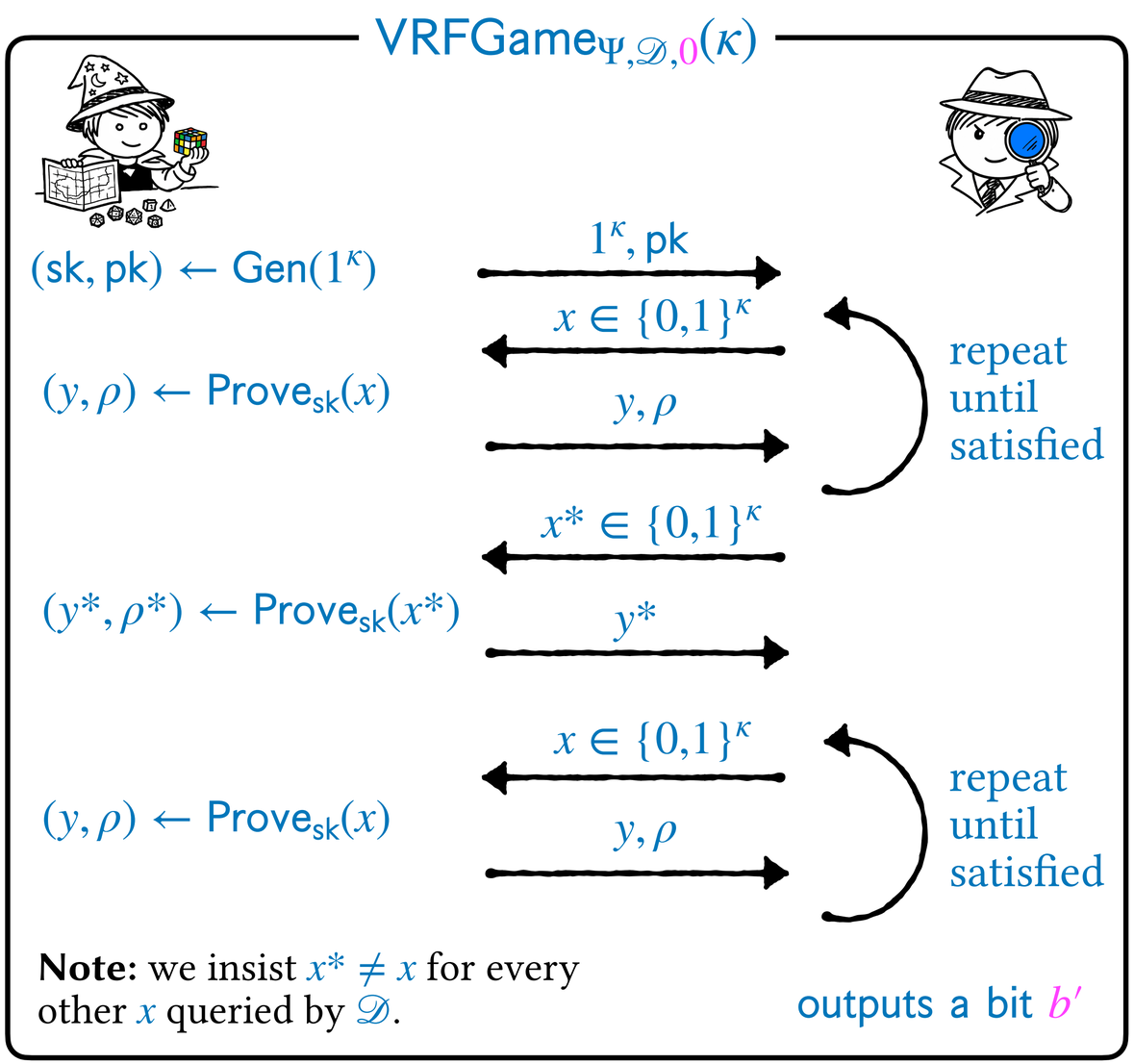
3. For every $(\mathsf{sk}, \mathsf{pk}) \in \mathrm{image}(\mathsf{Gen}(1^\kappa))$, $x \in \{0,1\}^\kappa$, and $(y, \rho) \in \mathrm{image}(\mathsf{Prove}_{\mathsf{sk}}(x))$, we have $\mathsf{Verify}_{\mathsf{pk}}(x, y, \rho) = 1$. This is the *correctness* condition. We say ρ is a *proof* for (pk, x, y).

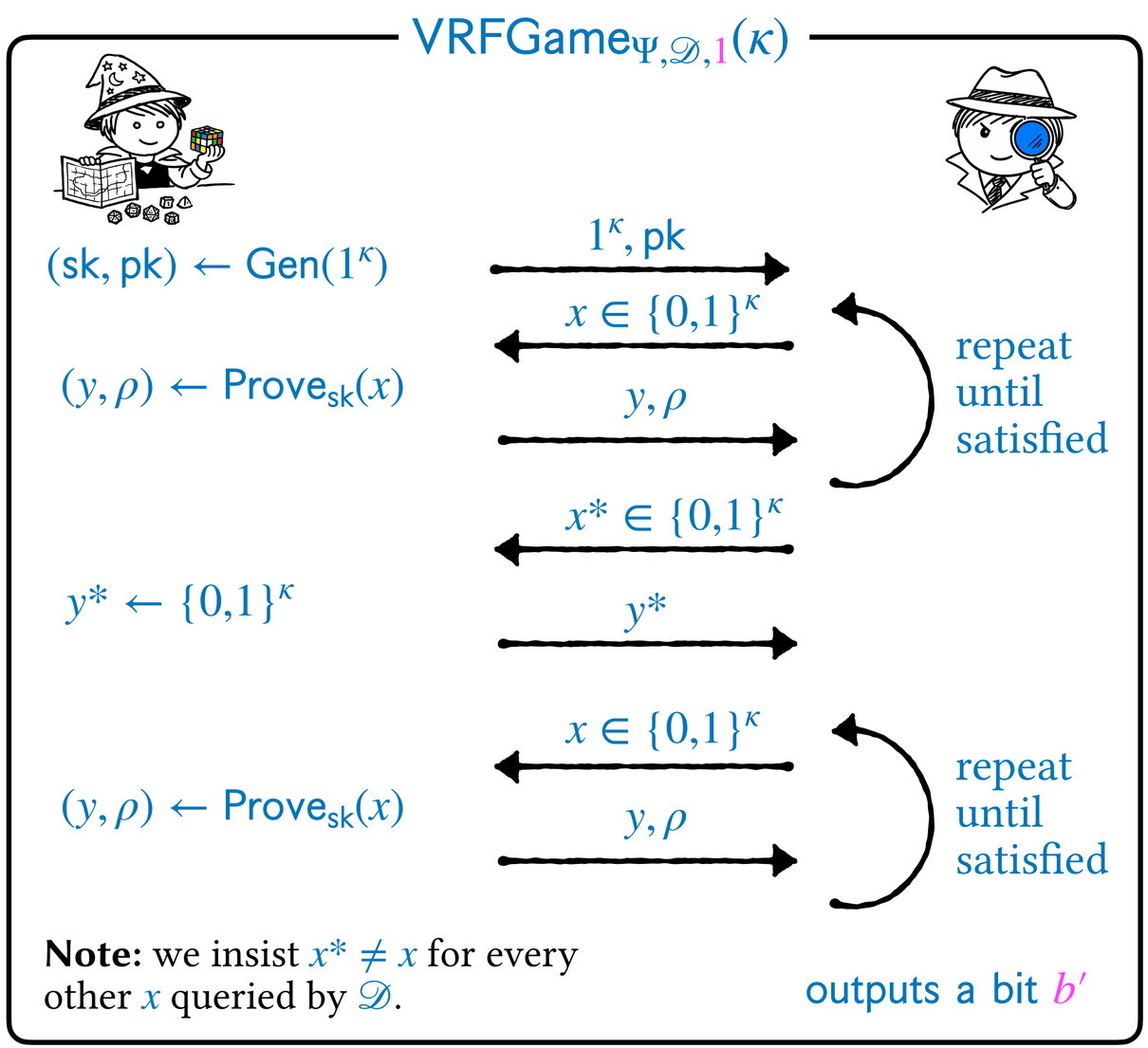
4. There is no tuple $(\mathsf{pk}, x, y, y', \rho, \rho')$ such that $y \neq y'$ but $\mathsf{Verify}_{\mathsf{pk}}(x, y, \rho) = \mathsf{Verify}_{\mathsf{pk}}(x, y', \rho') = 1$. In other words, for every (pk, x) there is a *uniquely* verifiable y. Notice that this means is a $F_{\mathsf{sk}}(x) = y : (y, \rho) \leftarrow \mathsf{Prove}_{\mathsf{sk}}(x)$ function!

5. $\forall$ PPT $\mathscr{D}$ $\exists$ a negligible function ε such that
$$\left| \Pr\left[\mathsf{VRFGame}_{\Psi,\mathscr{D},0}(\kappa) = 1\right] - \Pr\left[\mathsf{VRFGame}_{\Psi,\mathscr{D},1}(\kappa) = 1\right] \right| \leq \varepsilon(\kappa)$$

We know that ρ enables the distinguisher to determine whether a value y is truly random, or the output of $\mathsf{Prove}_{\mathsf{sk}}(x)$. Intuitively, we can ask for pseudorandomness *only* when ρ is withheld.

The VRF Game


VRFGame$_{\Psi,\mathscr{D},0}(\kappa)$
$(sk, pk) \leftarrow Gen(1^\kappa)$
$1^\kappa, pk$
$x \in \{0,1\}^\kappa$
$(y, \rho) \leftarrow Prove_{sk}(x)$
y, ρ
repeat until satisfied
$x^* \in \{0,1\}^\kappa$
$(y^*, \rho^*) \leftarrow Prove_{sk}(x^*)$
y^*
$x \in \{0,1\}^\kappa$
$(y, \rho) \leftarrow Prove_{sk}(x)$
y, ρ
repeat until satisfied
Note: we insist $x^* \neq x$ for every other x queried by $\mathscr{D}$.
outputs a bit b'


VRFGame$_{\Psi,\mathscr{D},1}(\kappa)$
$(sk, pk) \leftarrow Gen(1^\kappa)$
$1^\kappa, pk$
$x \in \{0,1\}^\kappa$
$(y, \rho) \leftarrow Prove_{sk}(x)$
y, ρ
repeat until satisfied
$x^* \in \{0,1\}^\kappa$
$y^* \leftarrow \{0,1\}^\kappa$
y^*
$x \in \{0,1\}^\kappa$
$(y, \rho) \leftarrow Prove_{sk}(x)$
y, ρ
repeat until satisfied
Note: we insist $x^* \neq x$ for every other x queried by $\mathscr{D}$.
outputs a bit b'

Micali's Consensus Protocol (2017)

Let $P_1, \ldots, P_n$ be parties of which at most $t < n/3$ are maliciously corrupted. Each P_i party starts with an input $x_i \in \{0,1\}$. We assume that they have already used a PKI to establish *trusted* public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ (that is, we assume the associated secret keys are all uniformly sampled) for some Verifiable Random Function Ψ.

Protocol $\pi_{\mathsf{Micali\text{-}Phase}}(r)$:

1. Each P_i sends x_i to everyone.
   a. If P_i receives 0 from $\geq n - t$ parties, then it sends 0^* to everyone (this is a special value that indicates "assume I will always send 0 from now on") and halts with output $(0, \top)$.
   b. If P_i receives 1 from $\geq n - t$, then it sets $v_i := 1$.
   c. If there is no $b \in \{0,1\}$ such that P_i receives b from $\geq n - t$ parties, then it sets $v_i := 0$.

Micali's Consensus Protocol (2017)

Protocol $\pi_{\text{Micali-Phase}}(r)$:

1. Each P_i sends x_i to everyone.
   a. If P_i receives 0 from $\geq n-t$ parties, then it sends 0^* to everyone (this is a special value that indicates "assume I will always send 0 from now on") and halts with output $(0,\top)$.
   b. If P_i receives 1 from $\geq n-t$, then it sets $v_i := 1$.
   c. If there is no $b \in \{0,1\}$ such that P_i receives b from $\geq n-t$ parties, then it sets $v_i := 0$.
2. Each P_i sends v_i to everyone.
   a. If P_i receives 1 from $\geq n-t$ parties, it sends 1^* to everyone and halts with output $(1,\top)$.
   b. If P_i receives 0 from $\geq n-t$, then it sets $w_i := 0$.
   c. If there is no $b \in \{0,1\}$ such that P_i receives b from $\geq n-t$ parties, then it sets $w_i := 1$.

Micali's Consensus Protocol (2017)

2. Each P_i sends v_i to everyone.
   a. If P_i receives 1 from $\geq n-t$ parties, it sends 1^* to everyone and halts with output $(1,\top)$.
   b. If P_i receives 0 from $\geq n-t$, then it sets $w_i := 0$.
   c. If there is no $b \in \{0,1\}$ such that P_i receives b from $\geq n-t$ parties, then it sets $w_i := 1$.
3. Each P_i computes $(\gamma_i, \rho_i) \leftarrow \mathsf{Prove}_{\mathsf{sk}_i}(r)$ and sends (w_i, γ_i, ρ_i) to everyone.
   a. If P_i receives 0 from $\geq n-t$ parties, then it outputs $(0,\perp)$.
   b. If P_i receives 1 from $\geq n-t$, then it outputs $(1,\perp)$.
   c. If there is no $b \in \{0,1\}$ such that P_i receives b from $\geq n-t$ parties, then:
      i. P_i computes $S_i := \{j \in [n] : \mathsf{Verify}_{\mathsf{pk}_j}(r, \hat{\gamma}_j^i, \hat{\rho}_j^i) = 1\}$, where $\hat{\gamma}_j^i$ and $\hat{\rho}_j^i$ are the values of γ_j and ρ_j respectively that were actually received by P_i from P_j.
      ii. P_i finds $k_i = \mathrm{argmin}_{j \in S_i}(\gamma_j^i)$ and outputs $(\mathrm{lsb}(\gamma_{k_i}^i), \perp)$.

A Security Sketch for Micali Consensus

Claim 1: If at the beginning of step 3, no honest party has halted and sent b^* for some $b \in \{0,1\}$, then with probability negligibly close to $1/3$ (or greater), the honest parties will output the same value at the end of step 3.

Proof Sketch:

- We know that every $F_{\mathsf{sk}_i}(x) = y : (y, \rho) \leftarrow \mathsf{Prove}_{\mathsf{sk}_i}(x)$ for $i \in [n]$ is a function, and that for any sk_i and x, no other y has a verifying proof ρ. It follows that if $\mathsf{Verify}_{\mathsf{pk}_j}(r, \hat{\gamma}_j^i, \hat{\rho}_j^i) = 1$ and $\mathsf{Verify}_{\mathsf{pk}_j}(r, \hat{\gamma}_j^h, \hat{\rho}_j^h) = 1$ (i.e. two honest parties P_i and P_h both accept the proof of P_j) then $\hat{\gamma}_j^i = \hat{\gamma}_j^h$ (i.e. P_j sent them both an honest evaluation of $F_{\mathsf{sk}_j}(r)$). This in turn implies that $\left| \bigcup_{i \in [n]} S_i \right| \leq n$.
- We have assumed that $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ are uniform (regardless of whether the corresponding parties are corrupted). In spite of the fact that the adversary can distinguish its *own* VRF evaluations from uniform strings, we can combine the pseudorandomness of the VRF with the result of the previous bullet to show that if $k_i := \operatorname{argmin}_{j \in S_i}(\gamma_j^i)$, then $\Pr[P_{k_i} \text{ is honest and agreed upon}] \gtrsim 2/3$. The proof of this is out of scope for lecture; think about it and see if you can figure it out!
- The above bullet is also where a bias-free property for VRFs can be used.

A Security Sketch for Micali Consensus

- We will call the event that P_{k_i} is honest and agreed upon "a good leader is elected." In this case, notice that we have $\text{lsb}(\gamma_{k_i}^i) = \text{lsb}(\gamma_{k_h}^h)$ for every pair of honest parties P_i and P_h. Magically, the good leader has already distributed an agreed-upon (pseudo)random bit.
- Assuming that a good leader is elected, we have three mutually exclusive cases:
  1. All honest parties receive $\geq n - t$ copies of some value $b \in \{0,1\}$ in step 3, and per 3a or 3b, they all output $(b, \perp)$.
  2. No honest parties receive $\geq n - t$ zeros or ones in steps 3, in which case, per 3c, they all output $(\text{lsb}(\gamma_k), \perp)$, for a value of $\text{lsb}(\gamma_k)$ that they agree upon (by the above bullet point).
  3. *One* group of honest parties receive $\geq n - t$ copies of a value $b \in \{0,1\}$ and per 3a or 3b, they output $(b, \perp)$, while *another* group of honest parties receive $\geq n - 2t$ copies of b and $\geq n - 2t$ copies of $1 - b$ (because the t corrupt parties behaved inconsistently). Since $\text{lsb}(\gamma_k)$ is agreed upon and indistinguishable from a uniform bit, the second group outputs $(b, \perp)$ with probability negligibly close to $1/2$.
- Thus, conditioned on a good leader, the parties agree at the end of Step 3 with probability $\gtrsim 1/2$. The overall probability that they are in agreement is thus $\gtrsim 1/2 \cdot 2/3 = 1/3$. ∎

A Security Sketch for Micali Consensus

Claim 2: If at the beginning of step 1, 2, or 3, the honest parties are in agreement about the value of x_i, v_i or w_i, then at the end of that step, they remain in agreement about the next value in the sequence and/or the output value (though they some may output $\perp$ and others $\top$).

Proof: If the honest parties are in agreement, then every honest party will receive $\geq n - t$ copies of the bit b that they agree upon. In all cases, they are then instructed to continue agreeing upon this value. ∎

Claim 3: If any honest party halts with output 0 (and sends 0^*) during step 1, then all honest parties agree on $v_i = 0$ at the end of step 1. Similarly, if any honest party halts with output 1 (and sends 1^*) during step 2, then all honest parties agree on $w_i = 1$ at the end of step 2.

Proof: If any honest party halts with output 0 in step 1, then they must have received $\geq n - t$ zeros. Because at most t parties are corrupt, the other honest parties must have received $\leq 2t$ ones and at $\geq n - 2t$ zeros. Thus by step 3c, they all set $v_i := 0$. The argument for step 2 and $w_i := 1$ is symmetric. ∎

Next, consider the *full* protocol π_{Micali} where the parties run the $\pi_{\text{Micali-Phase}}$ protocol repeatedly. They start with $r = 1$. Any party P_i that outputs $(b, \perp)$ from iteration r sets $r := r + 1$, and runs the protocol again with $x_i := b$ as their input.

A Security Sketch for Micali Consensus

When a party outputs $(b, \top)$ and halts, it does *not* run any more iterations of the protocol or send any more messages; in this case, it will have sent the message b^*, and all other parties will imagine that the halted party sends b for all future messages in all future iterations, without exception.

Claim 4: In each iteration of $\pi_{\text{Micali-Phase}}$ after the first one, all of the honest parties halt with probability at least $\gtrsim 1/3$. Furthermore, when they halt, they output some consistent $(b, \top)$.

Proof: Direct corollary from Claims 1 and 3, and the fact that $t < n/3$. ∎

Claim 4 implies that consistency will eventually be reached if the protocol runs for long enough. Combining Claim 4 with Claim 2, gives us *validity* for each phase individually (and thus validity for the whole protocol). It remains only to observe that each iteration of $\pi_{\text{Micali-Phase}}$ contains 3 rounds, and by Claim 4, all of the parties halt during the first two rounds of the fourth iteration, in expectation. Therefore, the expected number of rounds is ≤ 11. Thus we have…

Theorem 3: Assuming *trusted* PKI and *verifiable random functions*, there is a *randomized* BA protocol with expected-constant rounds.

So what about randomized BA/Broadcast when $t \geq 1/3$?

We have lower bounds, but our best constructions are not tight to those bounds (see Shi, Chapter 10).

There are still many important open problems. Sometimes it takes years to solve one. Maybe you could do it next :)

CS4501 Cryptographic Protocols
Lecture 22: DS Lower Bound, Randomized Broadcast

https://jackdoerner.net/teaching/#2026/Spring/CS4501