Blockchains & Distributed Ledgers

Lecture 06

Dimitris Karakostas

Slide credits: DK, Aggelos Kiayias, Nikos Leonardos

Permissionless Protocols

- Bitcoin and similar PoW-based blockchain protocols provide a **permissionless** setting:
  - Anyone can participate in the protocol and receive BTC as rewards by performing the PoW-based mining operation
- **Minting new coins** (via PoW) makes it feasible for anyone (possessing sufficient hashing power) to participate
- The ledger itself is public, readable and writeable by anyone
  - read (retrieve ledger information): connect to the network and download the ledger
  - write (insert new information to the ledger): obtain some bitcoins and create a transaction

Dynamic Availability

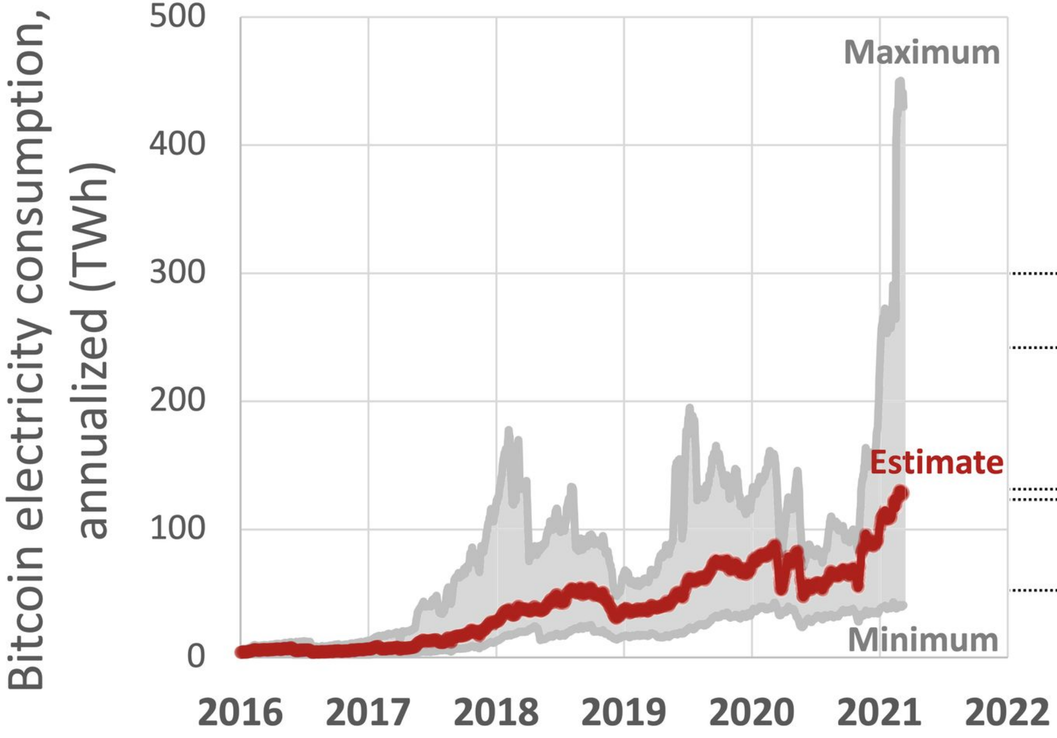
- Parties join and leave at will
- Need to bootstrap a chain when (re)joining):
  - Bitcoin's *"longest chain rule"* (most difficult chain)
- Number of online/offline parties changes over time
  - Analysis must account for that
- No *a priori* knowledge of participation levels
- Unannounced disappearance

Classic BFT protocols *do not* operate under general dynamic availability

Bitcoin’s Energy Problem

- Bitcoin resolves dynamic availability via PoW
  - Parties have limited access to a resource (computational power)
  - They repeatedly try to solve cryptographic puzzles (hashes)
  - A puzzle solution allows to create a block and append it to the chain
- Bitcoin is **extremely** energy **inefficient**
  - The used resource is physical
  - The hash-based lottery consumes extreme energy to ensure the protocol’s security
    - An energy arms race between the good guys and the (potential) bad guys
  - Bitcoin presumes that it is under attack *at all times*

Bitcoin's Energy Problem - electricity consumption



Country equivalent:

UK: 300 TWh

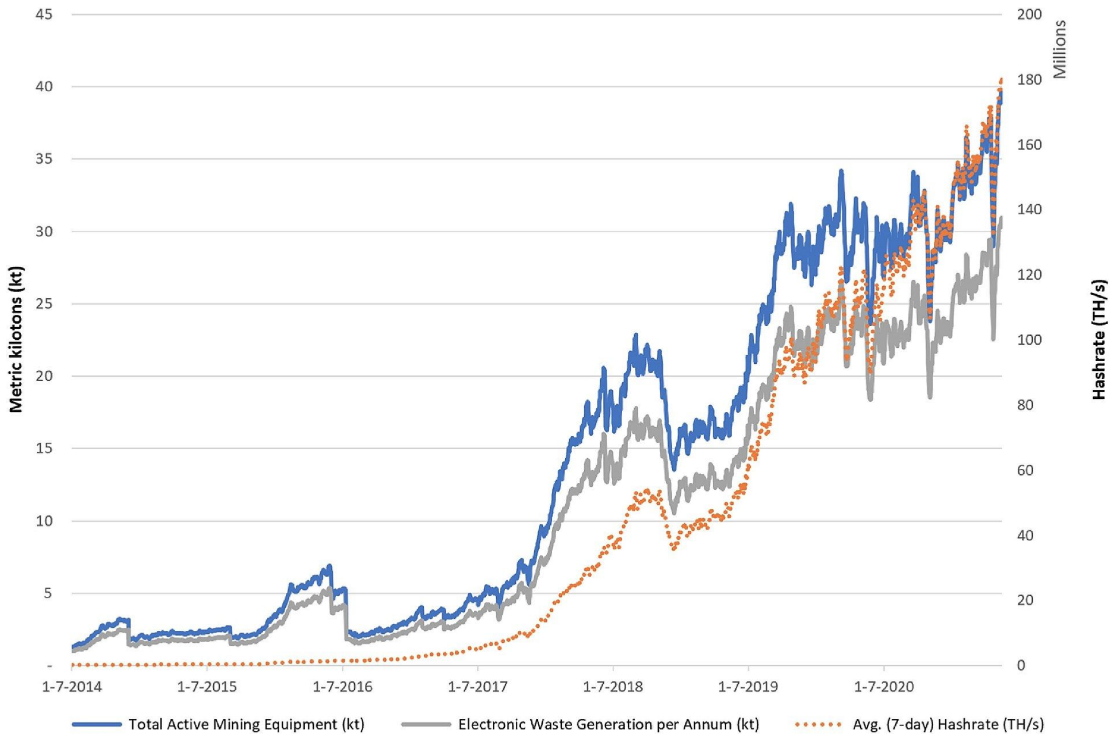
Spain: 242 TWh

Sweden: 131 TWh
Argentina: 125 TWh

Israel: 56 TWh

https://en.wikipedia.org/wiki/Environmental_impact_of_bitcoin

Bitcoin’s Energy Problem - electronic waste



https://en.wikipedia.org/wiki/Environmental_impact_of_bitcoin

Bitcoin's Energy Problem - "digital crude"

Between 2016-2021:

- **per coin climate damages** from BTC were **increasing**, rather than decreasing as the industry matured
- during certain time periods, BTC climate **damages exceed the price of each coin** created
- on average, each **$1 in BTC market value** created was responsible for **$0.35 in global climate damages**
  - between beef production and crude oil burned as gasoline
  - an order-of-magnitude higher than wind and solar power

[Benjamin A. Jones, Andrew L. Goodkind & Robert P. Berrens. Economic estimation of Bitcoin mining's climate damages demonstrates closer resemblance to digital crude than digital gold (2022)]

Proof-of-Stake (PoS)

The time slot

- Time is continuous
- Protocol breaks time in **slots**
  - Defines a “slot length” parameter (in seconds)
- Slot **large enough**
  - E.g., if network is assumed synchronous, slot length depends on graph’s diameter, s.t. all parties receive a message within a time slot
- Slot **not too large**
  - Otherwise protocol is slow
  - E.g., in Bitcoin waiting until tx is published is 10 mins (until a block is created) or more (for safety, k blocks are needed)
- Parties act based on the time slot they are in

Proof-of-Stake (PoS)

- **Sybil resilience** depends on “**stake**”
    - Stake: the amount of digital assets (tokens) a party controls
    - Akin to computational power in PoW, but stake is **digital**
    - **Energy efficient**: no need to consume high amounts of energy to run the stake-based lottery
- Parties produce blocks **proportionally** to the stake they control
    - Smallest rate: linearly proportional
- Assumption: **Adversary does not control a stake majority**
    - Corrupted parties control, on aggregate, less stake than the honest parties

Two broad categories:

- Nakamoto-style
- BFT-style

From PoW to PoS, Nakamoto style

The setting:

- The number of all assets is known
  - Tokens are recorded on the ledger
- The public key that controls each asset is known
  - Stake transfers (e.g., payments) are recorded on the ledger
- One block should be created per slot

High level idea:

- At each slot, choose one of the assets *at random*
  - Relaxation: choose a *very small* number of assets at random
- The owner of the chosen asset is eligible to produce a block at that slot
  - Owner: the person with the private key that owns the asset

PoS setting

- Assume (for now) that stake does not shift
  - There are no changes in stake ownership
  - The initial set of stake owners is known (e.g., hardcoded in the genesis block)
- Let n be a node
  - vk_n: the public key of n
  - $stake_n$: the stake owned by n
  - Both vk_n and $stake_n$ are known by all parties

Recall PoW

$$H(x, s, ctr) < T$$

- H: hash function
- ctr: PoW counter
- x: MTR of block’s transactions
- s: hash of parent block’s header
- T: difficulty threshold

From PoW to PoS - Attempt 1

- Require: $H(x, s, vk_n) < T \cdot stake_n$
    1. threshold is proportional to stake of each party (right side of inequality)

From PoW to PoS - Attempt 1

- Require: $H(x, s, vk_n) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)

Grinding Attack on x:

- Attacker can try different MTRs to find one that satisfies the inequality

From PoW to PoS - Attempt 2

- Require: $H(s, vk_n) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)

From PoW to PoS - Attempt 2

- Require: $H(s, vk_n) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality’s left side does not depend on MTR (to prevent grinding)

Stalling Hazard:

- With some probability (depending on T), no *vk* will satisfy the equation → No block is created at that slot → No parameter in the inequality changes → The protocol stalls

From PoW to PoS - Attempt 3

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)

From PoW to PoS - Attempt 3

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
    1. threshold is proportional to stake of each party (right side of inequality)
    2. inequality's left side does not depend on MTR (to prevent grinding)
    3. ts (e.g., timestamp) changes as slots change (to prevent stalling)

Content Malleability:

- Block's content (transactions) not represented in the header anymore → Attacker can alter the previous blocks' transactions without altering the headers (which are validated in the PoS mechanism)

From PoW to PoS - Attempt 4

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
    1. threshold is proportional to stake of each party (right side of inequality)
    2. inequality's left side does not depend on MTR (to prevent grinding)
    3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
    4. Have both the headers and the payloads form a chain (to prevent content malleability)
        - Headers contain a pointer to parent header
        - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload

From PoW to PoS - Attempt 4

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
    1. threshold is proportional to stake of each party (right side of inequality)
    2. inequality's left side does not depend on MTR (to prevent grinding)
    3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
    4. Have both the headers and the payloads form a chain (to prevent content malleability)
        - Headers contain a pointer to parent header
        - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload

Posterior Corruptions:

- Attacker can corrupt parties *after the slot passes* when they create a block → Attacker can change part (or all) of the chain's history

From PoW to PoS - Attempt 5

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
  4. Have both the headers and the payloads form a chain (to prevent content malleability)
     - Headers contain a pointer to parent header
     - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload
  5. Key Evolving Signatures (KES) (to prevent posterior corruptions)
     - Parties refresh their keys periodically (delete old keys, create new keys linked with old ones)

From PoW to PoS - Attempt 5

- Require: $H(s, vk_n, ts) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
  4. Have both the headers and the payloads form a chain (to prevent content malleability)
     - Headers contain a pointer to parent header
     - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload
  5. Key Evolving Signatures (KES) (to prevent posterior corruptions)
     - Parties refresh their keys periodically (delete old keys, create new keys linked with old ones)

Adaptive attack:

- vk that satisfies inequality is publicly known *before* the time slot starts → Attacker can predict the slot "leader schedule" → Can corrupt a party that is known to be leader of a specific future slot

From PoW to PoS - Attempt 6

- Require: $VRF(s, sk_n, ts) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
  4. Have both the headers and the payloads form a chain (to prevent content malleability)
     - Headers contain a pointer to parent header
     - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload
  5. Key Evolving Signatures (KES) (to prevent posterior corruptions)
     - Parties refresh their keys periodically (delete old keys, create new keys linked with old ones)
  6. Verifiable Random Function (VRF) (to prevent adaptive corruptions)
     - A party runs the inequality using its *secret* key
     - The VRF output is verifiable *publicly* (i.e., with the party's public key)

From PoW to PoS - Attempt 6

- Require: $VRF(s, sk_n, ts) < T \cdot stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
  4. Have both the headers and the payloads form a chain (to prevent content malleability)
     - Headers contain a pointer to parent header
     - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload
  5. Key Evolving Signatures (KES) (to prevent posterior corruptions)
     - Parties refresh their keys periodically (delete old keys, create new keys linked with old ones)
  6. Verifiable Random Function (VRF) (to prevent adaptive corruptions)
     - A party runs the inequality using its *secret* key
     - The VRF output is verifiable *publicly* (i.e., with the party's public key)

Nothing-at-stake:

- Each block offers different randomness; Creating uncle blocks has (practically) no cost → Grinding on different blocks of the tree (i.e., different *s*)

From PoW to PoS - Attempt 7

- Require: VRF(R_{epoch}, sk_n, ts) < T · $stake_n$
  1. threshold is proportional to stake of each party (right side of inequality)
  2. inequality's left side does not depend on MTR (to prevent grinding)
  3. ts (e.g., timestamp) changes as slots change (to prevent stalling)
  4. Have both the headers and the payloads form a chain (to prevent content malleability)
     - Headers contain a pointer to parent header
     - Payload (transactions) contains a pointer to: i) parent header; ii) parent payload
  5. Key Evolving Signatures (KES) (to prevent posterior corruptions)
     - Parties refresh their keys periodically (delete old keys, create new keys linked with old ones)
  6. Verifiable Random Function (VRF) (to prevent adaptive corruptions)
     - A party runs the inequality using its *secret* key
     - The VRF output is verifiable *publicly* (i.e., with the party's public key)
  7. Refresh randomness more periodically (to prevent nothing-at-stake)
     - Divide execution in *epochs* (of a specific amount of slots)
     - Each epoch's first block contains a (securely generated) randomness R_{epoch} for all slots in that epoch
     - This is the idea behind Ouroboros Praos

Dynamic Stake

- **Stake shifts** occur via payments
  - New stakeholders (i.e., keys) are added
  - The stake of old stakeholders is reduced
- Stake ownership distribution **depends on the chain** (i.e., branch)
  - Different blocks (in different chains) will contain different transactions

Key Grinding Attack

- Each user’s key is created locally (by the user)
- Creating keys is (effectively) costless
- If the randomness source can be biased, the attacker might generate multiple keys until they find one that favors them (w.r.t. the used randomness)

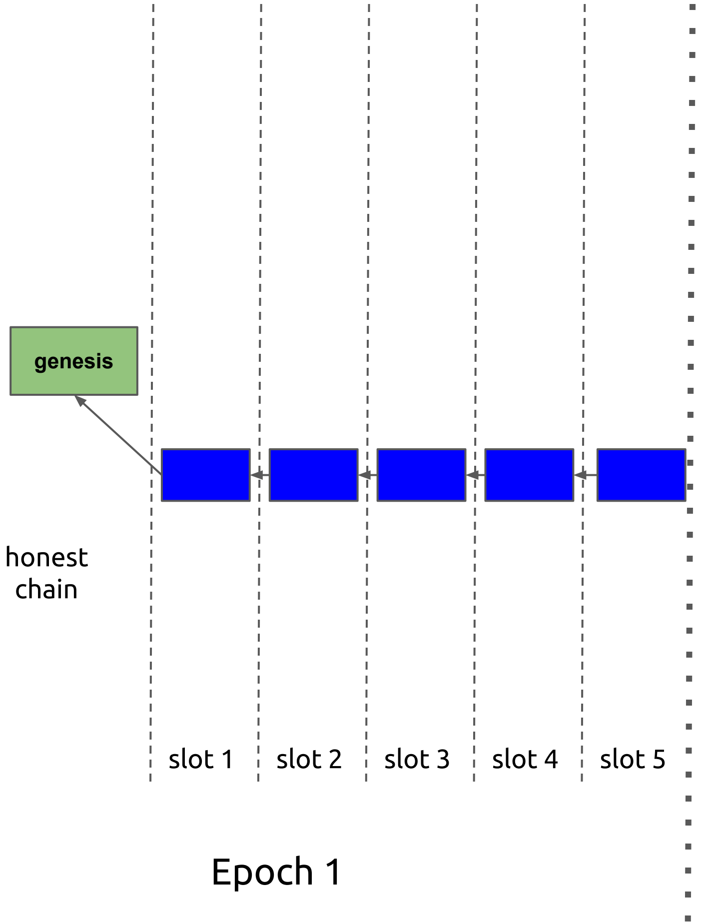
Key Grinding Attack

- Each user's key is created locally (by the user)
- Creating keys is (effectively) costless
- If the randomness source can be biased, the attacker might generate multiple keys until they find one that favors them (w.r.t. the used randomness)

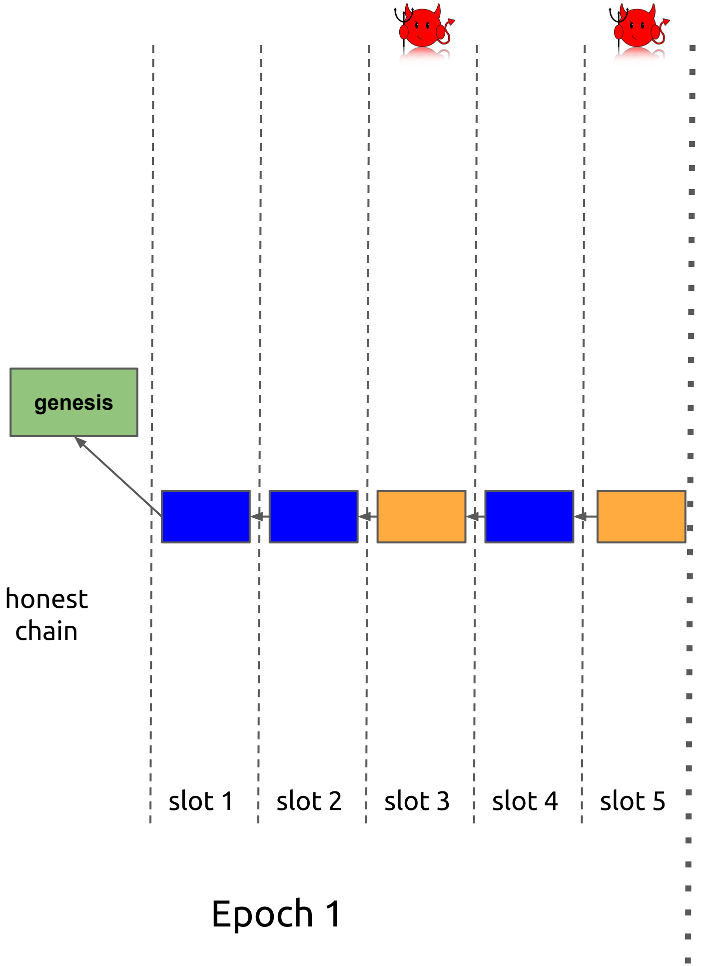
Solutions:

- Combine (possibly) adversarially-generated randomness with honestly-generated on (see Lecture 4)
- Get more randomness from little randomness
  - Hint: randomness extractors

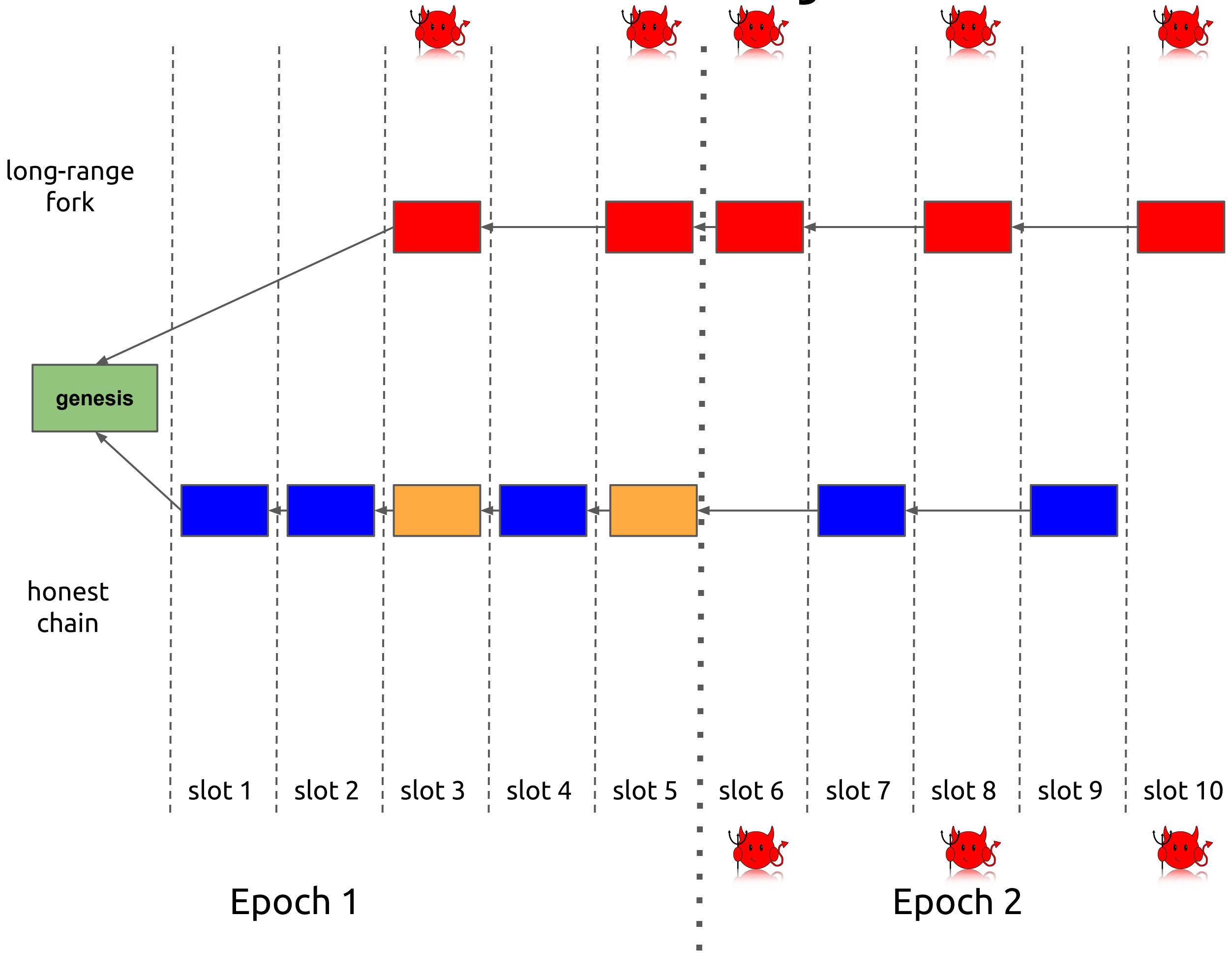
Long-range attack



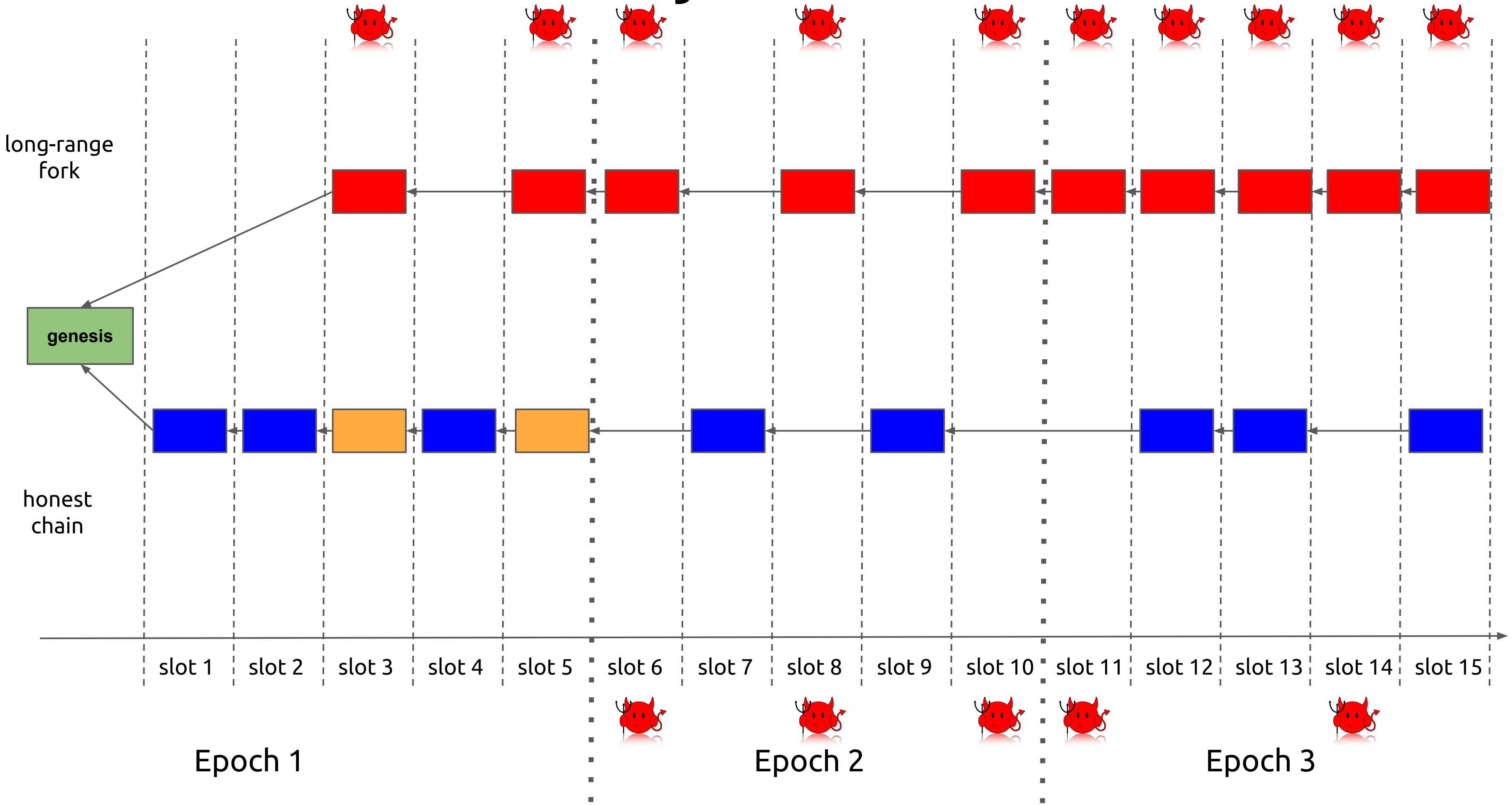
Long-range attack



Long-range attack



Long-range attack



Long-range attack

- The attack:
  - Starting from an old block, the attacker creates a chain of adversarial-only blocks
  - In this chain, it collects the rewards for every block
    - In this branch/“fork”, the attacker’s stake is increased
  - After some point, the attacker gets stake majority in this fork
  - Because creating blocks is costless, the attacker can create an arbitrarily long chain

Long-range attack

- The attack:
  - Starting from an old block, the attacker creates a chain of adversarial-only blocks
  - In this chain, it collects the rewards for every block
    - In this branch/“fork”, the attacker’s stake is increased
  - After some point, the attacker gets stake majority in this fork
  - Because creating blocks is costless, the attacker can create an arbitrarily long chain
- Solution 1: checkpoints
  - Basic idea: a checkpoint is a block that is **never dropped** by the user, even if they receive a longer chain without it
  - **Chain decision** prioritizes checkpoints over longest chains
  - Nodes have to go **online periodically** to retrieve the latest checkpoints
    - For any checkpoints issued while the node is offline, the node can be tricked by an adversary
  - Checkpoints have been used by Ouroboros, Snow White, and even Bitcoin (for other reasons)

Long-range attack

- The attack:
  - Starting from an old block, the attacker creates a chain of adversarial-only blocks
  - In this chain, it collects the rewards for every block
    - In this branch/“fork”, the attacker’s stake is increased
  - After some point, the attacker gets stake majority in this fork
  - Because creating blocks is costless, the attacker can create an arbitrarily long chain
- Solution 2: chain density
  - Basic idea: **immediately after the fork**, the **honest** chain’s blocks are **more “dense”** compared to the attacker’s (forked) chain
  - A new node that joins the system, chooses a path at each fork by following the most dense branch
  - The idea behind Ouroboros Genesis

BFT-style PoS

High level idea:

- At each slot, subselect a committee of stakeholders
  - Getting elected to the committee is proportional to the party's owned stake
- The committee runs a BFT protocol to agree on the new block
- Each block is *immediately finalized*
  - Liveness with parameter 1, no need to wait for k blocks
- The idea behind Algorand

BFT-style PoS

High level idea:

- At each slot, subselect a committee of stakeholders
  - Getting elected to the committee is proportional to the party's owned stake
- The committee runs a BFT protocol to agree on the new block
- Each block is *immediately finalized*
  - Liveness with parameter 1, no need to wait for k blocks
- The idea behind Algorand

Some security considerations:

- Where does the randomness come from (for the committee selection)?
- How to prevent grinding attacks?
- How to prevent adaptive corruptions?
- How to prevent long-range attacks?

Food for thought

- How to ensure that parties have a synchronised clock?
  - How do parties coordinate in terms of the time progress?
  - How do parties agree on which time slot is active at any point in time?
  - Hint: Ouroboros Chronos (PoS-based), Permissionless Clock Synchronization (PoW-based)
- Will rational parties delete their keys?
  - Key erasures (in KES) are necessary to prevent posterior attacks.
  - Do parties get any benefit by not deleting their keys? Can an attacker incentivize this?
- What happens if an attacker gets a majority during one epoch?
  - Can the system recover from temporary adversarial majority?
  - Can PoW systems recover?
  - Hint: Self-healing blockchains

Permissioned Ledgers

Permissioned Protocols

- Participation is **restricted**:
  - Producing transactions and/or blocks can only be performed after being authorized by (some) other nodes
- In the simplest case, the set of nodes is **static**:
  - the set of participating nodes is fixed and determined at the onset of protocol’s execution

Permissioning How-To

- Most straightforward approach:
    - employ a PKI (Public-Key Infrastructure)
- Use digital signatures / authentication protocols
- **Certificate authorities** can authorize other entities
    - authorization includes a signature from the CA on the entity's public-key, identity info etc
    - example: TLS/SSL
- Sharing certificate authority information is necessary
    - All computer systems come with preloaded certificates from certificate authorities - a **setup assumption**
- Certificates need to be **revoked** in case the corresponding secret keys become exposed or the algorithms used are not safe anymore

X.509 Certificates

- Internet standard since 1988
  - http://www.ietf.org/rfc/rfc3280.txt
- Hierarchical

Version
Serial Number
Algorithm / Parameters
Issuer
Period of Validity: not before date not after date
Subject
Algorithm/ Parameters/ Key
x509v3 extensions
...
Signature

X.509 does not specify cryptographic algorithms

Digital Signatures and Certificates

- A certificate contains a digital signature
- Cryptographic design of digital signatures involves typically:
  - A cryptographic signing operation that acts on a fixed input of a specific type and has a public-verifiability feature
  - A cryptographic hash function that takes arbitrary strings and maps them to the data type suitable for the signing operation
  - Common setting today: SHA2 with RSA or DSA

Secure channels and certificates

- Possession of mutually acceptable certificates:
  - permits authenticated communication (exchanging signed mechanism between two entities)
  - allows building a secure channel
- *TLS 1.3* can be used to build such secure channel:
  - Based on cryptographic protocols like Diffie-Hellman key exchange
  - Data confidentiality ensured

Static Permissioned Blockchain

- Prior to system's start:
  - the nodes register their certificates
  - these certificates are included in the genesis block
- Using these certificates, all nodes are capable of:
  - authenticating each participant
  - allowing interaction with the shared state, in a way prescribed by the participants' credentials
- The set of participants remains the same throughout the execution
- This is the simplest form of a PKI / public-key directory

A Centralised Permissioned Ledger

- Assume just a “LOG” of transactions
- **One of the participants** acts as a server and maintains the LOG
- Readers and writers to the LOG authenticate with the server and can perform read and write operations
- Consistency of the LOG is guaranteed, assuming the **server is trusted**
- Liveness of the LOG is guaranteed, assuming the server is **trusted and functional**
- If server is corrupted, the ledger is compromised

(The course’s testnet is built on a centralized permissioned ledger.)

Bitcoin Permissionless Ledger

- The genesis block contains no certificate information
- **Reading** from the LOG is **open**
  - anyone can do it, without credentials
- Writing to the LOG requires a specific type of credentials
  - write: insert data into the log
  - Nodes can obtain valid credentials (accounts) by generating a public and secret-key and:
    - mine a block (and be rewarded with BTC) or
    - buy BTC from another node
- Once the LOG records their account credit, they can issue transactions (and pay the necessary fees)
- In essence: crediting a bitcoin account is akin to creating a certificate that imparts the account holder with certain permissions w.r.t. the ledger

Distributed Permissioned Ledger

- A **number of servers** maintain the ledger (LOG) individually
- All share the **same genesis block** that identifies all participants
- Assuming a synchronous operation, at each round, readers and writers:
    - authenticate with the servers
    - interact with the LOG in a prescribed fashion

Distributed Permissioned Ledger

- A **number of servers** maintain the ledger (LOG) individually
- All share the **same genesis block** that identifies all participants
- Assuming a synchronous operation, at each round, readers and writers:
  - authenticate with the servers
  - interact with the LOG in a prescribed fashion
- Readers authenticate to each server and obtain read access
- Writers authenticate to each server and provide their inputs
- Servers run a **consensus protocol** to agree what inputs should be included in the LOG

Read Requests

- Is it possible to restrict read requests, as in the centralized setting?
  - Hint: Nodes keep blocks of transactions private and issue them only to authenticated users
- TLS can be used to build a secure channel between the reader and the responding node
- Requirement that all servers remain honest (as they all share the LOG)
- Is is possible to impose read restrictions on servers as well?
  - Hint: Threshold signatures

Reader/Writer Management

- Readers and writers can authenticate to each server referring to the information in the genesis block
- It is possible to introduce additional readers and writers by suitably issuing certificates to other users
- Note that each participant would then need to show a valid certificate chain, that establishes their privileges for the requested read or write access

BFT Protocol Example

“Classical” BFT Consensus (example)

- Focus on write requests: we want to ensure LOG liveness and consistency
- We will build a “byzantine fault tolerant” (BFT) agreement protocol that uses two important tools:
  - a graded broadcast
  - a binary consensus protocol

Graded Consensus

- Parties involved :
  - a single **sender**
  - several receivers
- The i-th receiver outputs (M_i, G_i)
  - M_i: the output message
  - $G_i \in \{0, 1, 2\}$: the grade of the message

Properties

- If the sender is **honest**, then $M_i = M_j$ for all i, j and $G_i = 2$
- If the sender is **malicious** and one receiver outputs (M, 2), then all other honest receivers output (M, G_i) with $G_i \in \{1, 2\}$

Graded Broadcast Protocol

<u>**Communication**</u>

- **Round 1.** The sender sends the message *M* to all receivers
- **Round 2.** The *i*-th receiver, who obtained $M_{1,i}$ in round 1, sends it to all receivers
- **Round 3.** The *i*-th receiver, who obtained $M_{2,j,i}$ from the *j*-th receiver in round 2:
    - if there is a single message that was sent by at least *2n/3* receivers, it sends it to all receivers
    - else does nothing

Graded Broadcast Protocol

Communication rounds

1. The sender sends the message M to all receivers
2. The i-th receiver, who obtained $M_{1,i}$ in round 1, sends it to all receivers
3. The i-th receiver, who obtained $M_{2,j,i}$ from the j-th receiver in round 2:
    - if there is a single message that was sent by at least $2n/3$ receivers, it sends it to all receivers
    - else does nothing

Output Generation

The honest i-th receiver does the following:

- If a single message was received from *at least* $2n/3$ receivers in round 3, output that message as M_i and set $G_i = 2$
- If a single message was received from *at least* $n/3$ receivers in round 3, output that message as M_i and set $G_i = 1$
- In any other case, output *fail* as M_i and set $G_i = 0$

Graded Broadcast Protocol (Analysis: $t < n/3$)

Theorem #1

If the sender is honest and broadcasts M, then all *honest* receivers P_i will output $G_i = 2$ and M in the output generation stage.

Proof

- If the sender is honest, then all honest receivers will receive the same message M in round 1.
- Since $t < n/3$, each receiver will receive M *at least* $2n/3$ times in rounds 2 and 3 (from the honest parties).

Communication rounds

1. The sender sends the message M to all receivers
2. The i-th receiver, who obtained $M_{1,i}$ in round 1, sends it to all receivers
3. The i-th receiver, who obtained $M_{2,j,i}$ from the j-th receiver in round 2:
   - if a single message was sent by at least $2n/3$ receivers, send it to all receivers

Output Generation

The honest i-th receiver:

- If a single message was received from *at least 2n/3* receivers in round 3, outputs that message as M_i and set $G_i = 2$

Graded Broadcast Protocol (Analysis: t < n/3)

Lemma #1

If two honest receivers send a message in round 3, it *must be* the same.

Proof

Suppose an honest party P sends message M in round 3:

1. P has received M by at least 2n/3 parties in round 2 *(by definition)*
2. Let **h** be the number of *honest parties* that sent M in round 2: $h \geq (2n/3) - t > n/3$ *(by assumption)*
3. Let **p** be the parties *capable* of sending a *different* message $M' \neq M$ in round 2: $p = n - h < 2n/3$ *(by step 2, i.e., since h honest parties sent M)*
4. Therefore, any other honest party in round 3 will send M or do nothing

Communication rounds

1. The sender sends the message M to all receivers
2. The *i*-th receiver, who obtained $M_{1,i}$ in round 1, sends it to all receivers
3. The *i*-th receiver, who obtained $M_{2,j,i}$ from the *j*-th receiver in round 2:
   - if there is a single message that was sent by at least *2n/3* receivers, it sends it to all receivers
   - else does nothing

Graded Broadcast Protocol (Analysis: $t < n/3$)

Theorem #2

Suppose the i-th receiver returns $G_i = 2$ and a message M_i; for the j-th honest receiver's output (M_j, G_j), it holds $M_i = M_j$, $G_j \in \{1, 2\}$.

Proof

First, we show that it cannot be that $M_j = fail$:

1. The i-th party received M_i from at least 2n/3 receivers in round 3 (of which at most t<n/3 adversarial)
2. So, *more than* n/3 *honest* parties sent M_i in round 3

Now, suppose $M_j \neq M_i$:

1. M_j was sent by at least n/3 receivers in round 3 *(by definition)*
2. At least one of them is honest *(since $t < n/3$)*
3. By Observation #2, it holds $M_i = M_j$ [*contradiction*]

Communication

Round 3. The *i*-th receiver, who obtained $M_{2,j,i}$ from the *j*-th receiver in round 2:

- if a single message was sent by at least *2n/3* receivers, send it to all receivers

Output Generation

The honest *i*-th receiver:

- If a single message was received from *at least 2n/3* receivers in round 3, outputs that message as M_i and set $G_i = 2$
- If a single message was received from at least n/3 receivers in round 3, output that message as M_i and set $G_i = 1$
- In any other case, output *fail* as M_i and set $G_i = 0$

From Graded Broadcast to a BFT-Ledger

Graded broadcast *is not enough*:

- If grade $G_i = 1$, party P_i cannot know if other honest parties received the message

A simplistic approach:

- execute n/3 phases (to guarantee at least one honest sender encountered)
- in each phase:
  - A designated sender organizes all valid transactions as *M* and performs a graded broadcast
  - A binary consensus protocol determines whether everyone's grade is 2 or not:
    - If true, each node signs the output to generate a public endorsement and appends *M* to their LOG (together with the signatures)
    - otherwise, LOG remains the same

Byzantine Binary Consensus

- (RECALL) n parties, t adversarial
- $v_i \in \{0, 1\}$ the input of party i
- Honest parties should *decide* on values $u_i \in \{0, 1\}$ satisfying the following properties:
  - **Termination**: values u_i are well defined for all honest parties
  - **Agreement**: if parties i and j are honest, then $u_i = u_j$
  - **Validity**: if, for every honest party i, there exists $v \in \{0, 1\}$ such that $v_i = v$, then each honest party i outputs $u_i = v$

Note: We examine the *synchronous* setting

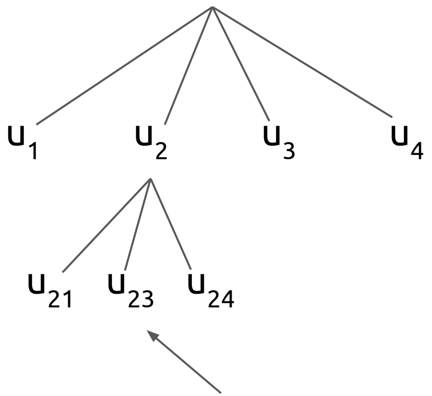
Exponential Information Gathering Algorithm (EIG)

Algorithm Sketch:

- At round 1, send everyone your input
- At round r+1, send everyone all messages you received at round r (avoiding redundant messages)

Each party arranges the messages in its own EIG tree:

- Let u_1,...,u_n be the messages received in the first round (including one's self)
- $u_{12\ldots k}$ is the value v s.t. *(i_k told i) that (i_{k-1} told i_k) that … that (i_1 told i_2)* that i_1's initial value was v



u_{23}: The value party 3 told me that party 2 sent them in the previous round.

(Food for thought) What is the size of the tree?

EIG Termination

The EIG algorithm terminates after t+1 rounds. The output value of each party is defined as follows:

- For each leaf v in the EIG tree, set $z_v = u_v$
- For an internal node v, set z_v equal to the majority of the z-values of its children; if the majority is not defined, set $z_v=0$ (without loss of generality)
- Define the output as z_{root}

(Food for thought) Prove that EIG satisfies: i) agreement; ii) validity. (Hint)

Impossibility results - asynchronous setting

- *Theorem [LSP1982]*: Impossible for $n < 3t + 1$.
- *Theorem [FL1982]*: Impossible in t rounds.
  - *Example:* The EIG algorithm with $t = 1$ needs at least 2 rounds:
    - If a party received a single 1, its output should be 0. (Because the 1 could be coming from the adversary.)
    - If a party received two 1s, its output should be 0. (Because one of them could have been sent from the adversary, while another party could have received a single 1 and will decide 0 according to the previous statement.)
    - And so on… (by induction, the output will always be 0, contradicting validity)
- *Theorem[GM1998]*: Doable for $n > 3t$ in $t + 1$ rounds.
- *Theorem [DS83]*: Doable for $n > 2t$ assuming a PKI.

Impossibility results - asynchronous setting

- *Theorem [BT1985]*: Asynchronous Byzantine Consensus is impossible with $n < 3t + 1$, even if the parties have agreed on a PKI (setup).
  - Partition parties into sets A, B, C of size at most t and consider 3 scenarios:
    i. A malicious, B and C honest with inputs 0. The adversary sends no messages. The honest parties should decide on 0 until some time T_A.
    ii. B malicious, A and C honest with inputs 1. The adversary sends no messages. The honest parties should decide on 1 until some time T_B.
    iii. C malicious, B and A honest with inputs 0 and 1 respectively. The adversary communicates with B as the honest C in scenario A and with A as the honest C in scenario B. At the same time every communication between A and B is delayed for time at least $\max\{T_A, T_B\}$.
  - The crux is that A has the same view in scenarios B and C. Similarly for B, in scenarios A and C. Agreement in scenario C is impossible, if validity is achieved in scenarios A and B.