Blockchains & Distributed Ledgers

Lecture 06

Aggelos Kiayias

Slide credits: AK, Nikos Leonardos

Lecture 06 Overview

1. The Permissioned and Permissionless setting
2. Digital signatures and certificates
3. BFT = Byzantine fault tolerance. A classical BFT protocol for a permissioned ledger
   a. Graded Consensus
   b. Binary Byzantine Agreement
4. Proof of Stake Protocols

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.
- The mechanism of pouring currency in the system via proof of work, makes it feasible for anyone (possessing sufficient hashing power) to participate.
- The ledger itself is public, readable and writeable by anyone (the latter assuming one possesses bitcoin)

Permissioned Protocols

- Participation is restricted:
- Producing transactions and/or blocks can only be performed after being authorized by the other nodes.
- In their simplest form the set of nodes is **static**: the set of nodes implementing the protocol is fixed and determined at the onset of protocol execution.

Permissioning How-To

- Most straight approach: employ a PKI (=public-key infrastructure).
- Based on digital signatures / authentication protocols.
- **Certificate authorities** can authorize other entities.
- authorization includes a signature from the CA on the entity’s public-key, identity information etc.
- Sharing certificate authority information is necessary; (how, where?)

X.509 Certificates

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

Structure of x.509

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.
- Recall that 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.

Certification considerations

- All computer systems come with preloaded certificates from certificate authorities. This provide a **setup assumption**.
- Certificates need to be revoked in case the corresponding secret keys become exposed or the algorithms used are not safe anymore.
- In a blockchain system, certificate information can be provided as part of the genesis block.

Secure channels and certificates

- Possession of mutually acceptable certificates not only permits authenticated communication (exchanging signed mechanism between two entities) but also allows building a secure channel
- Protocol **TLS 1.3** is used to build such secure channel.
  - It relies on cryptographic protocols such as the Diffie Hellman key exchange. It can ensure the confidentiality of the data exchanged.

Static Permissioned Blockchain

- All participants are identified by self-signed certificates in the genesis block.
- The set of participants remains the same throughout the execution.
- This is the simplest form of a PKI / public-key directory.

Permissioning

- Prior to system operation the nodes register their certificates that are included in the genesis block.
- Using such certificates, all the nodes are capable of authenticating each participant and allowing interaction with the shared state in a way that is prescribed by the participants’ credentials.

A Centralised Permissioned Ledger

(let’s focus on just a “LOG” of Transactions)

- One of 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.

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 can only be done in specific ways (issuing transactions).
- Nodes can obtain valid credentials (accounts) by generating a public and secret-key and either mining a block (which will reward their account with BTC) or buy BTC from another node.
- Once the LOG records their account credit, they can issue transactions.
- In essence: crediting a bitcoin account is 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.
- Each share the same genesis block that identifies all participants.
- Assuming a synchronous operation, at each round, Readers and Writers authenticate with the servers and interact with the LOG in a prescribed fashion.

Distributed Permissioned Ledger, II

- 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.

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 her privileges for the specific read or write access that is requested.

Read Requests

- Is it possible to restrict read requests as in the centralized setting?
- Nodes can maintain blocks of transactions private and issue them only to users that are authenticated.
- The TLS protocol can be used to build a secure channel between the reader and the responding node.
- Note that the above would require that all servers remain honest (as they all share the LOG).

“Classical” BFT Consensus

- Focus on write requests next. 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 and several receivers.
  - The i-th receiver outputs (M_i, G_i).
  - The value G_i is in {0,1,2}.
  - 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 other honest receivers output (M,G_i) with Gi>=1.

Graded Broadcast Protocol, 1

- **Round 1.** The sender sends the message *M* to all receivers.
- **Round 2.** The *i*-th receiver obtains *M1i* from round 1 and sends it to all receivers.
- **Round 3.** The *i*-th receiver obtains *M2ji* from the *j*-th receiver in round 2 and performs the following:

    if there is a single message that was sent by at least *2n/3* receivers then send it to all receivers. Else do nothing.

Graded Broadcast Protocol, 2

Output Generation. The *i*-th receiver obtains $M3ji$ from the *j*-th receiver in round 3.

- If there is a single message that was sent by at least $2n/3$ receivers output that message as Mi and set $Gi=2$.

- If there is a single message that was sent by at least n/3 receivers output that message as Mi and set $Gi=1$.

- In any other case output *fail* and $Gi=0$.

Graded Broadcast Protocol, 3

Analysis. Assume that malicious parties are $t < n/3$.

Observation #1

If the sender is honest, then each receiver will receive the same message >= $2n/3$ times in rounds 2 and 3. As a result all honest receivers will output $G_i=2$ and that message in the output generation stage.

Graded Broadcast Protocol, 4

Observation #2. 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. Then P has received *M* by at least 2n/3 parties. Given this, observe that $2n/3-t$ > n/3 honest parties have sent *M* in round 2.

Thus < n-n/3 = 2n/3 parties are capable of sending a different message than *M* in round 2. Hence another honest party P’ in round 3 will send either *M* or nothing.

Graded Broadcast Protocol, 5

Observation #3

Suppose the *i*-th receiver returns $Gi=2$ and let Mi be the message it generates. For the *j*-th receiver's output (Mj, Gj) it holds $Mi=Mj$, $Gj>0$.

The *i*-th receiver has received the message Mi from at least $2n/3$ receivers in round 3.

=> More than $n/3$ honest receivers have sent Mi in round 3. Thus it cannot be that Mj=fail.

Now suppose Mj is not equal to Mi.

Then we deduce Mj was sent by at least n/3 receivers in round 3. => at least one of them is honest. Thus by Observation #2 it holds $Mi=Mj$.

From Graded Broadcast to a BFT-Ledger

A simplistic approach: execute n/3 phases to guarantee an honest sender will be encountered. In each phase perform:

- A designated sender organizes all valid transactions it collected as **M** and performs a graded broadcast.
- A binary consensus protocol determines whether everyone's grade is 2 or not. If that is true each node signs the output to generate a public endorsement and appends **M** on their LOG (together with the signatures). Otherwise LOG remains the same.

Byzantine Binary Consensus

(RECALL) n parties (1,2,...,n), t adversarial.

Let $v_i \in \{0,1\}$ be the input of party i.

Honest parties should *decide* on values $u_i \in \{0,1\}$ satisfying the following properties.

- **Agreement**: if parties i and j are honest, then $u_i = u_j$.
- **Validity**: if there exists $v \in \{0,1\}$ such that $v_i = v$ for each honest party i, then $u_i = v$ for each honest party i.
- **Termination**: values u_i are well defined for all honest parties.

Note: We will examine the synchronous setting

Exponential Information Gathering Algorithm (EIG)

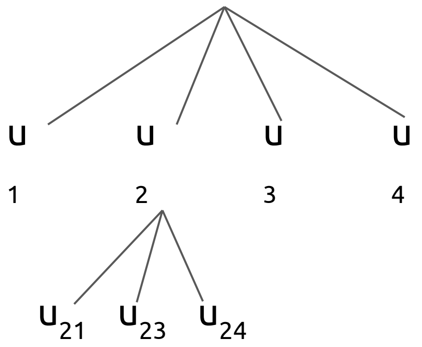
Algorithm Sketch.

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

Each party arranges the messages in its own EIG tree.

- Let $u_1,\ldots,u_n$ be the messages received in the first round. (including itself)
- Subsequently, u_{xj} is the value received from j as the value u_x in j's tree.

Note: there need be no repetitions in the label of a node (e.g., x in u_x should contain distinct identifiers).



The value party 3 told me that party 2 send him in the previous round.

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}.

(Im)possibility results - synchronous 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.

1. If a party received a single 1, its output should be 0. (Because the 1 could be coming from the adversary.)
2. 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.)
3. 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.

Proof Partition parties into sets A, B, C of size at most t. Consider 3 scenarios.

A. 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.
B. 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.
C. 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.

A blockchain related to proof-of-stake

Servers $S_1,\ldots,S_n$ with shared verification keys $pk_1,\ldots,pk_n$ and private signing keys $sk_1,\ldots,sk_n$.

B_0: Genesis block containing the public info.

$B_i = (k,d,sl,\sigma_{sl},\sigma_{block})$, where

k: hash of previous block

d: data

sl: slot number

σ_{sl}: signature on sl by $S_{sl \bmod n}$

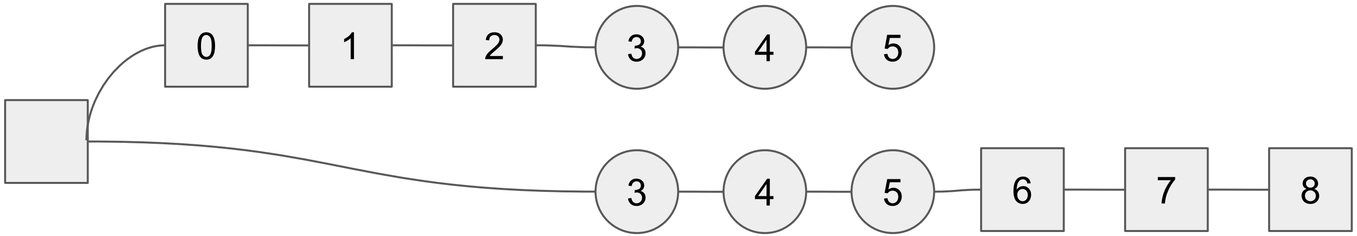
σ_{sl}: signature on whole block by $S_{sl \bmod n}$

At any step, each server, extends the longest known blockchain

Characteristic sequences and executions

The characteristic sequence of an execution with L slots 0,1,...,L-1 is a binary string w in $\{0,1\}^L$ such that $w_i=1$ iff $S_{i \bmod n}$ is adversarial.

Example w = 000111000. Squares denote honest slots (including the genesis) and circles the adversarial ones. The adversary keeps the chain ending with slot 5 on top hidden so that S_6 extends the bottom chain. This results in a **fork,** two disjoint chains (except for the genesis block) of maximum length. The corresponding characteristic sequence w is called **forkable.**



Forkable sequences

Fact If weight(w)<length(w)/3, then w is not forkable.

Proof Let n=length(w) and t=weight(w) and assume w is forkable. The two chains must have length at least n-t (prove formally by induction). Thus, 2(n-t) is a lower bound on the sum of their lengths. On the other hand, the n-t honest parties have contributed at most n-t, while the adversarial parties have contributed at most 2t (because a given chain contains a slot at most once). Thus, (n-t)+2t is an upper bound on the sum of their lengths. We have

$$2(n-t) \leq (n-t) + 2t \Longrightarrow n \leq 3t.$$

Consensus inspired from proof-of-stake [KR2018]

- Fix an arbitrary ordering of the servers: $S_1,\ldots,S_n$.
- Construct a blockchain for 5t+2 rounds, recording your own input bit as data in any block you create.
- Upon termination output the majority of the first 2t+1 blocks of your chain.

Theorem If $n > 3t$, the protocol satisfies agreement and validity.

Proof [KR2018] It can be shown that the first 2t+1 blocks are common to all honest parties; this implies agreement. Validity follows from the fact that among the first 2t+1 at most t are adversarial and so the majority of them belong to honest parties.

Bitcoin Consensus

- Miners run the Bitcoin protocol recording their own input bit as data in any block they compute.
- When their chain has at least 2k blocks (for some security parameter k), the broadcast it and stop.
- Output is the majority of the bits recorded in the first k blocks.

Theorem [GKL15] If $t<n/3$, the above protocol satisfies Agreement and Validity with probability $1-e^{-\Omega(k)}$.

Remark For Agreement, $t<n/2$ suffices.

Common-Prefix Property and Agreement

Common-Prefix Property For any pair of honest parties adopting the chains C_1 and C_2 at rounds r_1 and r_2 respectively. If $r_1 \leq r_2$, then $C_1[-k]$ is a prefix of C_2, where $C_1[-k]$ is C_1 without its last k blocks.

Common-Prefix Property implies Agreement. This is because the parties are pruning at least k blocks from their chains when keeping only their initial k blocks. Thus, the initial k blocks are common to all honest parties.

In [GKL2015] it is shown that in Bitcoin the Common-Prefix fails with probability exponentially small in k, if the adversary's hashing power is sufficiently bounded below ½. It follows that Agreement is satisfied with probability $1-e^{-\Omega(k)}$, when t is sufficiently less than n/2.

Chain-Quality Property and Validity

Chain-Quality Property (informal) Among any sufficiently large number of consecutive blocks in an honest party's chain, a fraction of at least (n-2t)/(n-t) have been computed by honest parties.

Chain-Quality implies Validity, when t is sufficiently less than n/3. This is because, in that case, the majority of the first k blocks have been computed by honest parties. Thus, if all honest parties have input v, the majority of values recorded in the first k blocks will be v.

In [GKL2015] it is shown that in Bitcoin Chain-Quality fails with probability exponentially small in k, if the adversary's hashing power is sufficiently bounded below ½. It follows that Validity is satisfied with probability $1-e^{-\Omega(k)}$, when t is sufficiently less than n/3.

Dynamic Availability

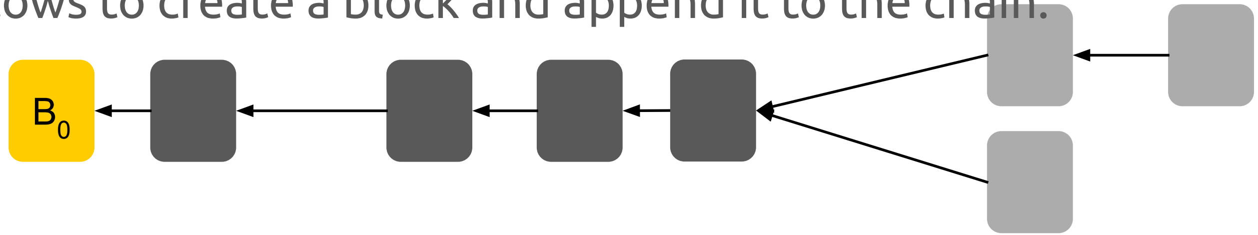
Recall: **Dynamic availability (DA).**

- Parties join and leave at will. They need to bootstrap a chain when (re-) joining.
  - Easy in Bitcoin: “**longest-chain rule”** (general: most difficult chain).
- Number of online/offline parties changes over time
  - Analysis must account for that -
- There is no *a priori* knowledge of participation levels is required by the protocol.
- Unannounced disappearance.

● Classical BFT protocols do not operate under general **dynamic availability**.

Bitcoin Energy Disadvantage

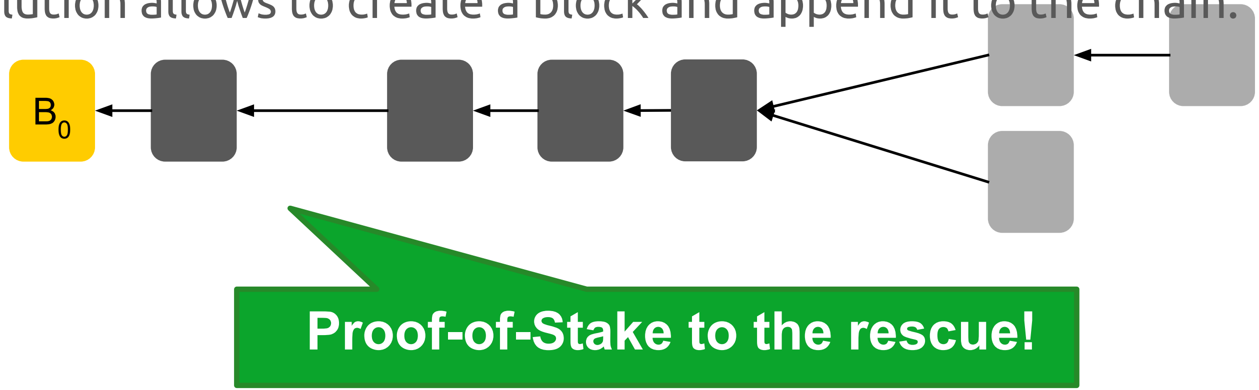
- **Bitcoin Proof of Work Mechanism:**
  - Parties repeatedly try to solve cryptographic puzzles. A solution allows to create a block and append it to the chain.



  - **Bitcoin is not energy efficient** as the hash-based lottery consumes a lot of energy to ensure the protocol's security.
  - Classical BFT protocols are much more energy efficient.

Proof of Stake

- **The case of Bitcoin:**
  - Parties repeatedly try to solve cryptographic puzzles. A solution allows to create a block and append it to the chain.



Proof of Stake

Proof-of-Stake Blockchains:

- Use **stake** (a virtual resource) instead of hashing power.
- Miners = **Stakeholders**.
- Next **stakeholder** to produce block **elected with probability proportional to stake.**

Proof of Stake

Proof-of-Stake Blockchains:

- Use Stake (a virtual resource) instead of hashing power.
- Miners = Stakeholders.
- Next stakeholder to produce block elected with probability proportional to stake.

Two categories:

Nakamoto-style consensus (e.g., the Ouroboros protocol - KRDO16)

BFT-style consensus (e.g., Algorand - CM16)

Realizing the Ledger

Complications of PoS vs. PoW:

- 📫 **PoS has costless simulation**:
  No physical resources to create blocks: several transaction histories could be generated “in the adversaries head”.
- 📫 **Long-Range attacks in the threat model**:
  Adversary tries to deceive (new) participants into believing the “wrong” history (which are cheap to generate).

References

[BT1985] Bracha, Toueg. Asynchronous consensus and broadcast protocols.

[FL1982] Fischer, Lynch. A lower bound on the time to assure interactive consistency.

[GKL2015] Garay, Kiayias, Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications.

[GM1998] Garay, Moses. Fully polynomial Byzantine agreement for n>3t processors in t+1 rounds.

[KRDO2016] Kiayias, Russell, David, Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol.

[LSP1982] Lamport, Shostak, Pease. The Byzantine generals problem.

[CM2016] Chen, Micali, Algorand a secure and efficient distributed ledger.