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

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
  - how?
  - where?

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

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
  - 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

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

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

Read Requests

- Is it possible to restrict read requests, as in the centralized setting?
- Nodes can 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)

“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)
- $G_i \in \{0, 1, 2\}$
- 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 all other honest receivers output (M, G_i) with $G_i \in \{1, 2\}$

Graded Broadcast Protocol (Communication)

- **Round 1.** The sender sends the message *M* to all receivers
- **Round 2.** The *i*-th receiver obtains $M_{1,i}$ from round 1 and sends it to all receivers
- **Round 3.** The *i*-th receiver obtains $M_{2,j,i}$ 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 (Output Generation)

The honest *i*-th receiver:

- If there is a single message received from *at least 2n/3* receivers in round 3, output that message as M_i and set $G_i = 2$
- If there is a single message 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: $t < n/3$)

Observation #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).

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

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:

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; it holds that $h \geq (2n/3) - t > n/3$
3. Let **h’** be the parties *capable* of sending a message $M' \neq M$ in round 2; it holds that $h' = n - h < 2n/3$
4. Therefore, any other honest party in round 3 will send either *M* or nothing

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

Observation #3

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 receiver received M_i from at least 2n/3 receivers in round 3
2. So, *more than* n/3 honest receivers 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. Therefore, at least one of them is honest (since t < n/3)
3. By Observation #2, it holds $M_i = M_j$ (contradiction)

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 an honest sender will be encountered)
- in each phase:
  - 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 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, 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)

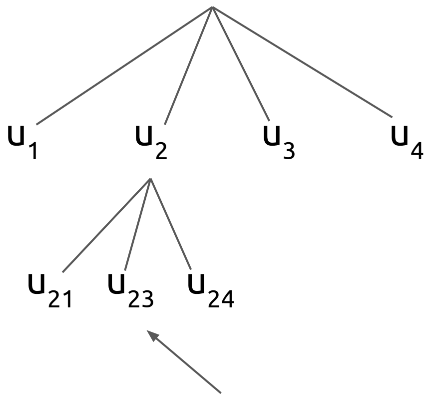
Algorithm Sketch:

- 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 itself)
- Subsequently, u_{xj} is the value received from j as the value u_x in j's tree.

Note: there need to 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 sent 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}

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.

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 is required by the protocol
- Unannounced disappearance

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

Bitcoin’s Energy Problem

- Bitcoin resolves dynamic availability via Proof-of-Work
  - 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 highly 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 bad guys
- Classical BFT protocols are *much more* energy efficient

Proof-of-Stake (PoS)

- In Proof-of-Stake systems:
  - Parties have limited access to a resource (digital coins, aka stake)
  - A party is elected to participate proportionally to their stake
  - An elected participant can create a block and append it to the chain
- PoS **is energy efficient**
  - The used resource is digital
  - No need to consume high amounts of energy to run the stake-based lottery

Proof-of-Stake (PoS)

- In Proof-of-Stake systems:
  - Parties have limited access to a resource (digital coins, aka stake)
  - A party is elected to participate proportionally to their stake
  - An elected participant can create a block and append it to the chain
- PoS **is energy efficient**
  - The used resource is digital
  - No need to consume high amounts of energy to run the stake-based lottery

Two categories:

- Nakamoto-style consensus (e.g., Ouroboros [KRDO16])
- BFT-style consensus (e.g., Algorand [CM16])

Implementing the ledger

Complications of PoS vs. PoW:

- 📫 PoS has **costless simulation:**
  - No physical resources to create blocks
  - several transaction histories can be generated “in adversary’s head”
- 📫 **Long-Range attacks**:
  - A new party P tries to join and decide which chain to choose
  - Adversary tries to deceive P into believing a “wrong” history (which is 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.