Blockchain Technologies 1

Assignment - 1

Aibek

2025-12-09

Contents

Module 1: Introduction to Blockchain Technology

1. Distributed vs. Centralized Ledgers

Explain how a distributed ledger differs from a centralized ledger in terms of trust, confidentiality, fault-tolerance, and attack surface. Provide at least 3 real-world examples for each.

Answer:

The essential discrepancy between distributed ledger (DL), commonly represented by blockchain technology, and centralized ledger (CL), usually a traditional database, is their architecture and the mechanisms they use for data integrity and coordination. A centralized system has clients that are connected to a single central server governed by an administrator, while a distributed ledger network is based on the peer-to-peer model where control as well as data distribution occur across many nodes.

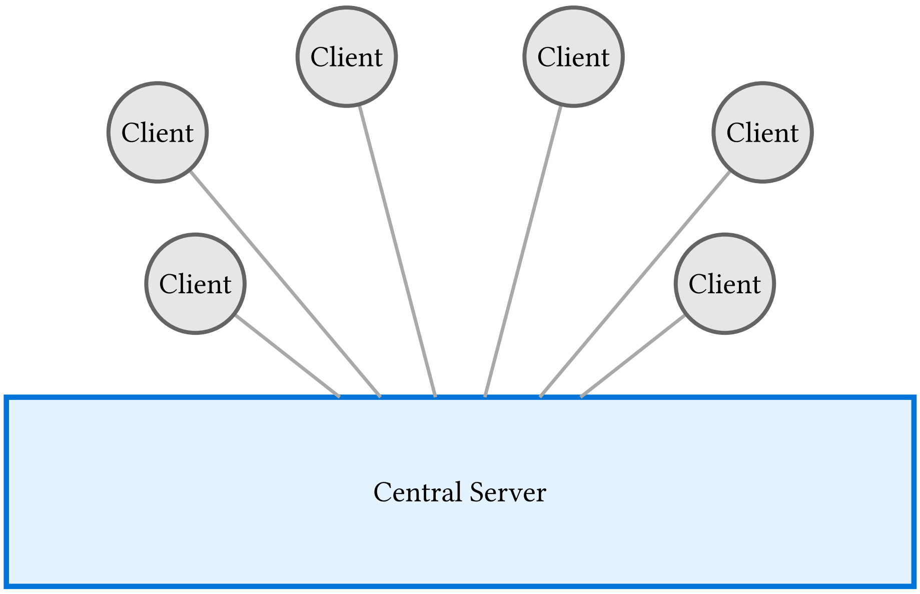
Trust

Centralized Ledger:

- **Trust Model**: Centralized systems rely on a trust-based model. Trust is implicitly or explicitly placed in a single central authority, administrator, or intermediary (like a bank) who manages the entire system and controls the data
- **Technical Reasoning**: The integrity and authenticity of the ledger are maintained exclusively by this single entity, meaning there is no technical guarantee against malicious actions by the controller

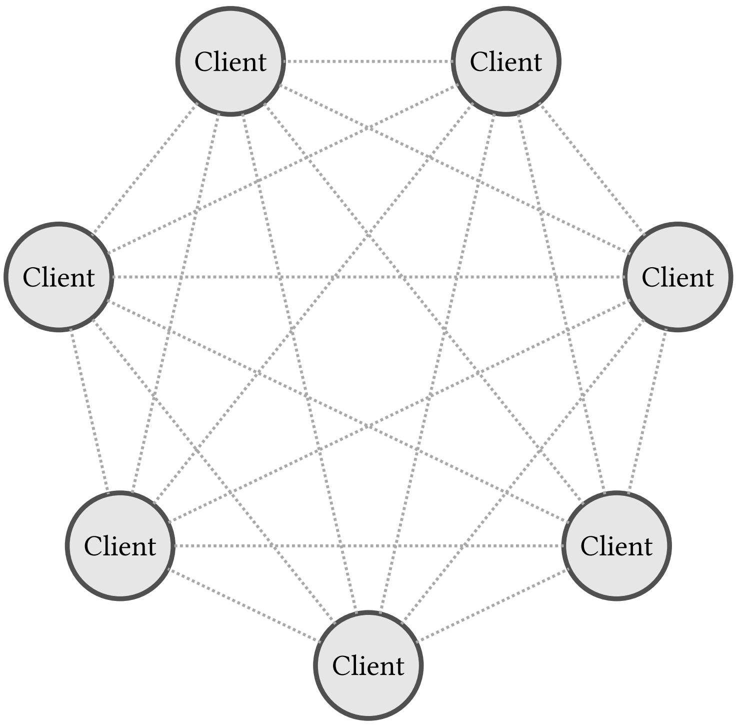
Distributed Ledger:

- **Trust Model**: Distributed ledgers operate on a trustless model (or distributed trust). Trust is established and maintained through cryptographic security and a consensus mechanism rather than relying on a single third party,,. Participants collectively agree on the state of the network.
- **Technical Reasoning:** Consensus protocols (like Proof-of-Work or Byzantine Fault Tolerance variants) ensure that new records are added only if participants collectively agree to do so. Transparency, where all participants possess the same verifiable information, reinforces this trust among participants



Centralized Ledger (CL): Single Point of Control

Figure 1: Diagram: Centralized Control



Decentralized Ledger (DL): No Single Point of Control

Figure 2: Diagram: Decentralized / Peer-to-Peer Network

Confidentiality

Centralized Ledger (CL):

- **Data Access:** The central authority is responsible for maintaining confidentiality and the access control policies established by it are the means through which this is done. The data is so to speak owned by the controlling entity, and access is limited to a few select individuals.
- **Technical Reasoning:** The central server's data security utilizes traditional methods such as authentication, access control, and physical security measures.

Distributed Ledger (DL):

- **Data Access:** Confidentiality is a big issue and very different for each type of DL:
  - Public (Permissionless) DLs has the main concern of transparency, thus all transactions are recorded in a shared, publicly available digital ledger. The user identities are usually anonymous but the transaction data is generally public.
  - Private/Consortium (Permissioned) DLs have the access limited only to the known participants, thus providing both confidentiality and transparency only within that group.
- **Technical Reasoning:** Cryptography is the main source of security for confidentiality and uses methods like encryption to the communication links (in transit) or the data when it is stored (at rest)

Fault-tolerance

Centralized Ledger (CL):

- **Resilience:** Centralized systems have very low fault tolerance as it is based on the one and only central server. This is exactly what a single point of failure means.
- **Technical Reasoning:** When the only central node ceases to function, the whole database becomes user-unreachable. The system is stuck because there is no distributed copy or coordinated failover, which is not the case since the core design is not supporting it.

Distributed Ledger (DL):

- **Resilience:** High fault-tolerance is the very property of distributed ledgers. The reason is that the data being distributed over many nodes, and the system is still available and on-going if one or more nodes go down.
- **Technical Reasoning:** Replication is the way to bring about fault tolerance. Typically, distributed systems that require consensus operate based on strict thresholds to guarantee safety, such as $N \geq 3F+1$ for tolerating Byzantine faults (where N is total nodes and F is faulty nodes). The system achieves liveness as long as a majority or a sufficient quorum of non-faulty nodes remains up and running and continues to process requests

Attack Surface

Centralized Ledger (CL):

- **Vulnerability:** The entire attack surface is concentrated in one single point-the central authority.
- **Technical Reasoning:** In total, an invader aspiring to disrupt, corrupt, or freeze the system will only have to take over the central node (the database, API endpoints, or the governing administrator)together with the system. Once the point of failure is breached, the hacker has full control of the data and system operations,.

Distributed Ledger (DL):

- **Vulnerability:** The network's overall security is based on the fact that all participating nodes are the points of attack. Through cryptographic hashing, the data is secured so that a chain is formed which is almost impossible to alter and which has very high resistance to penetration through tampering.
- **Technical Reasoning:** An adversary who wants to execute a devastating attack must first break the economic barriers and seize the control of a considerable part of the network which

usually means a supermajority or more than 50% of the computational power (in Proof of Work networks like Bitcoin) or over one-third of the total validator stake/nodes (in BFT/PoS networks). Changing a record is practically impossible because its hash is linked to consecutive blocks, hence altering one record requires recalculating the hashes of all subsequent blocks, making data tampering-proof.

Feature	**Centralized Ledger**	**Distributed Ledger**
Trust	Relies on a single authority	Trust is distributed among nodes
Confidentiality	High (Owner controls access)	Varies (Public vs. Permissioned)
Fault-tolerance	Low (Single point of failure)	High (Redundancy)
Attack Surface	Central server is the target	Consensus mechanism/Nodes

Real-World Examples:

Ledger Type	**Trust**	**Confidentiality**	**Fault-Tolerance**
Centralized	Traditional Banking Systems (like SWIFT, single bank ledgers)	Corporate ERP Systems (data managed by the company)	Traditional Web2 Applications (for example, a service running on one cloud server/database)
Distributed	Bitcoin (Public, permissionless network based on Proof-of-Work)	Ethereum (Public DApp platform using smart contracts and consensus)	Consortium Blockchains (like supply chain solutions involving several organizations using permissioned networks)

2. Definition of Immutability

Provide a rigorous technical definition of immutability. Explain how hash functions contribute to this and describe one scenario where immutability fails.

Answer:

Immutability in a blockchain refers to an unalterable and non-removable transaction after it has been confirmed and included in a block that belongs to the canonical chain. The ledger is only for adding new transactions: newly processed blocks can only be added one after another, and the entire history is kept permanently. Mistakes are not rectified by replacing them in the past, but rather with the addition of new reversing transactions.

How Hash Chaining Ensures Immutability The link between the blocks is established with the cryptographic hash of the preceding block:

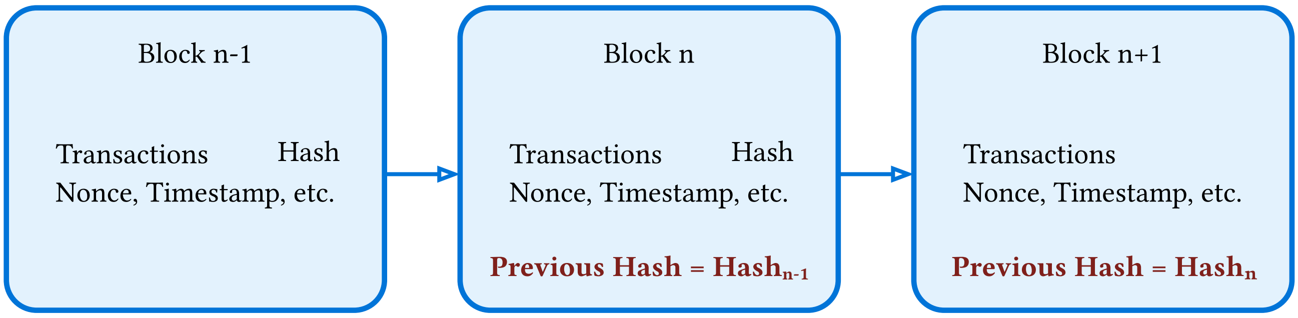
An alteration of even 1 bit in a past block dramatically changes its hash (avalanche effect).

This leads to the "previous hash" reference in the next block being broken.

To cover up the tampering, a hacker needs to perform the same intense computing work that involves re-mining the compromised block plus all the consecutive blocks.

In a Proof-of-Work system, the massive power cost needed makes this practically impossible.

Diagram illustration



Blockchain Hash Chaining – Enforcing Immutability

Figure 3: Each block stores the hash of the previous block (in red).
Modifying any block changes its hash, breaking the chain.

Immutability isn't absolute—it's economic finality. Suppose an adversary gains control of the hash power of the network that is more than 50%.

They first make the transaction public and then create a hidden chain that does not include that transaction and is longer than the public one. When the hidden chain is revealed, the participants in the network apply the longest-chain rule and accept it. The time of the original transaction is then erased from the canonical history ⟶ successful double-spend.

In this way, the immutability of the blockchain is guaranteed by cryptography but the costs that have to be borne for acquiring majority hash power finally determine the security.

3. Transparency vs. Privacy

Evaluate blockchain transparency vs. privacy. Compare Bitcoin vs. Ethereum and explain mixers, stealth addresses, and ZK proofs.

Answer:

4. DApp Architecture

Define DApp architecture in detail. Describe interactions between Smart Contracts, Off-chain backend, Frontend, Wallets, and Nodes.

Answer:

Replace with: DApp Architecture Diagram
(Use `#image("filename.png")`)

Figure: DApp Architecture Diagram

Component Interactions:

- **Smart Contract Layer:** …
- **Wallets:** …

Module 2: Cryptography Fundamentals

1. SHA-256 Computations

Compute SHA-256 hashes using Node.js, an online tool, and Linux terminal.

Answer:

1. Node.js Output:

```
Paste your Node.js output here
```

2. Linux Terminal Output:

```
Paste your sha256sum output here
```

Comparison:

2. Collision Resistance

Demonstrate collision resistance by modifying one bit of input. Explain the avalanche effect and the birthday paradox.

Answer:

Original Hash: `[Paste Hash]`

Modified Hash: `[Paste New Hash]`

The hashes are drastically different because...

Probability of Collision: Using the Birthday Paradox formula:

Module 3: Developer Environment Setup

Activity Requirements

Set up a blockchain environment (Node.js, npm, VS Code). Initialize a project and install `web3`, `ethers`, `crypto-js`.

Answer:

Package Installation & Version Check:

📷
Replace with: Screenshot of Terminal (node -v, npm -v) and package.json
(Use `#image("filename.png")`)

Figure: Screenshot of Terminal (node -v, npm -v) and package.json

Package Explanation:

- **web3:** …
- **ethers:** …
- **crypto-js:** …

Module 4: Models & Architecture

1. Bitcoin's UTXO Model

Draw a diagram of UTXO flow. Explain script validation, parallelism, and stateless validation.

Answer:

📷
Replace with: UTXO Flow Diagram
(Use `#image("filename.png")`)

Figure: UTXO Flow Diagram

Script Validation Steps:

1. …
2. …

2. Ethereum's Account Model

Explain EOA vs. Contract Accounts. Describe nonce, balance, storage, codeHash. Provide a JSON example.

Answer:

Account State JSON Example:

3. Security Implications

Analyze Replay vulnerabilities, Transaction malleability, Double-spend handling, Smart contract attack surface, and State bloat.

Answer:

4. EVM Architecture

Explain EVM bytecode execution, Stack-based model, Gas metering, and Error handling.

Answer:

5. Smart Contracts

Explain gas cost model and storage. Why are writes expensive? Provide code optimization example.

Answer:

Inefficient Code:

Optimized Code:

Explanation: …

Module 5: Wallets & Transactions

Step 1: Wallet Installation

Install MetaMask/Trust/Rabby. Submit screenshot of interface and public address.

Answer:

📷

Replace with: Wallet Interface Screenshot

(Use `#image("filename.png")`)

Figure: Wallet Interface Screenshot

Public Address: `0x...`

Step 2: Transaction Inspection

Decode a raw Bitcoin or Ethereum transaction.

Answer:

Selected Blockchain: Ethereum/Bitcoin

Transaction Hash: `0x...`

Decoded Fields:

Field	Value / Explanation
Nonce	...
Gas Price	...
Gas Limit	...
Value	...

Data/Input	...
To	...

Logic Explanation: