1-4

# 1. Introduction to Blockchain

## What is Blockchain?

* **Definition**: A blockchain is a **decentralized, distributed ledger** that records transactions across a network of computers. It consists of a chain of **blocks**, each containing data, a **hash** (unique fingerprint), and the **hash of the previous block**.
* **Key Features**:
  + **Decentralized**: No single entity controls the network.
  + **Immutable**: Once data is recorded, it cannot be altered.
  + **Transparent**: All participants can view the transactions.
  + **Secure**: Uses cryptography to secure data.

## Why Do We Need Blockchain?

* **Problems with Traditional Databases**:
  + Centralized systems are vulnerable to:
    - **hacks**,
    - **data manipulation**
    - **single points of failure**.
  + Lack of transparency and trust in centralized systems.
  + **No single point of failure**—improves security and reliability.
* **Blockchain Solves These Issues**:
  + Decentralization ensures no single point of control.
  + Immutability prevents tampering with data.
  + Transparency builds trust among users.

| **Feature** | **Traditional Database** | **Blockchain** |
| --- | --- | --- |
| Centralization | Centralized server | Decentralized (P2P) |
| Security | Can be hacked easily (vulnerable to single point attacks) | Uses cryptographic security |
| Modification | Data can be altered | Data is immutable |
| Trust | Requires trusted third-party | Trustless system |
| Transparency | Limited to authorized users | Transparent to all network participants |

# 2. How Blockchain Works

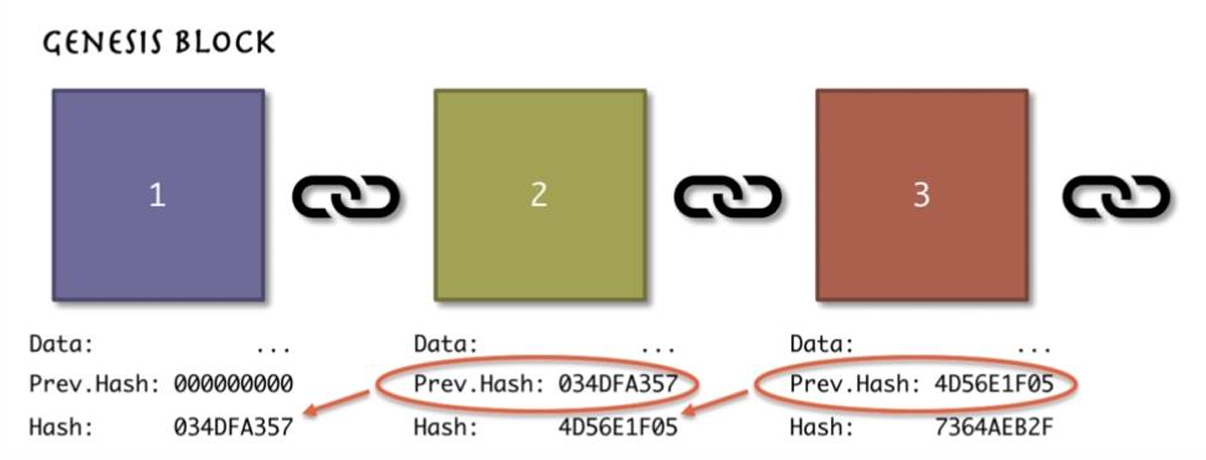
## Structure of a Block

* Each block contains:
  + **Data**: Transactions or other information (e.g., "Hello World").
  + **Previous Hash**: The hash of the previous block (links blocks together).
  + **Hash**: A unique fingerprint of the block’s contents (generated using **SHA256**).

### Genesis Block:

The first block in the blockchain, which has no previous hash.

#### **Diagram Explanation:**



* The diagram shows a simple blockchain with three blocks:
  + **Block 1 (Genesis Block)**: No previous hash.
  + **Block 2**: Contains the hash of Block 1.
  + **Block 3**: Contains the hash of Block 2.
* This chaining of blocks ensures **data integrity**.

# 3. SHA256 Hash Algorithm

## What is SHA256?

* **Definition**: A cryptographic hash function that takes an input and produces a **64-character hash**.
* **Key Properties**:
  + **Deterministic**: Same input always produces the same hash.
  + **One-Way**: Cannot reverse-engineer the input from the hash/ Cannot be reversed.
  + **Avalanche Effect**: A small change in input drastically changes the hash.
  + **Collision Resistant**: Two different inputs cannot produce the same hash.

#### **Example:**

* If you hash the word "Hello", you get a specific hash. If you change it to "hello" (lowercase), the hash will be completely different.

## **4. Immutable Ledger**

### **What is an Immutable Ledger?**

* **Definition**: A ledger that cannot be altered once data is recorded.
* **Traditional Ledgers**:
  + Prone to **tampering** and **errors**.
  + Example: Property deeds can be forged or destroyed.
* **Blockchain Ledger**:
  + Data is **immutable** and **secure**.
  + Example: Property records on a blockchain cannot be altered, ensuring ownership rights.

## **5. Peer-to-Peer (P2P) Network**

### **What is a P2P Network?**

* **Definition**: A decentralized network where each participant (node) has equal authority and a copy of the blockchain.
* **Key Features**:
  + **No Central Authority**: No single point of control.
  + **Anonymity**: Users interact without revealing their real identities.
  + **Consensus**: Majority of the nodes must agree on the validity of transactions.

#### **Explanation:**

* Each node has a copy of the blockchain.
* If a block is added, all nodes update their copies.
* If a hacker tries to alter a block, the network detects the inconsistency and rejects the change.

## **6. Mining and Consensus Mechanisms**

### **What is Mining?**

* **Definition**: The process of validating transactions and adding them to the blockchain.
* **How Mining Works**:
  + Miners solve a **cryptographic puzzle** to find a **nonce** (a number used once) that generates a hash below a certain target.
  + The first miner to solve the puzzle gets to add the block and is rewarded (e.g., with cryptocurrency).
  + Miners continuously change the nonce until they find a hash below the **difficulty target**.

#### **Nonce and Cryptographic Puzzle:**

* **Nonce**: A random number that miners change to generate a valid hash.
* **Target**: A threshold set by the network. Miners must find a hash below this target.
* **Golden Nonce**: The correct nonce that produces a valid hash(below the target).

## **7. Key Components of Blockchain**

### **1. Nodes:**

* Participants in the network that validate transactions and maintain the blockchain.

### **2. Miners:**

* Special nodes that solve cryptographic puzzles to add blocks, and get financial incentives based on their work.

### **3. Users:**

* Individuals or entities that perform transactions on the blockchain.

### **4. Smart Contracts:**

* Self-executing contracts with predefined rules (not covered in detail in this PDF but important to note).

## **8. Why is Blockchain Secure?**

### **1. Decentralization:**

* No single point of failure.

### **2. Cryptography:**

* Uses SHA256 hashing and public-private key encryption.

### **3. Consensus Mechanisms:**

* Proof of Work (PoW) and Proof of Stake (PoS) ensure agreement among nodes.

### **4. Immutability:**

* Once data is recorded, it cannot be altered.

### **9. Forging the Blockchain**

* A hacker would need to modify the blockchain on **51% of the nodes** to alter a transaction.
* The system automatically restores the correct version if an attack is detected.

## **10. Key Terms to Remember**

* **Block**: A container for data in a blockchain.
* **Hash**: A unique fingerprint of data.
* **Nonce**: A number used once in mining.
* **Genesis Block**: The first block in a blockchain.
* **P2P Network**: A decentralized network of nodes.

Forgery

## **1. Immutability Through Cryptographic Hashing**

### **What is Hashing?**

* Each block in the blockchain contains a **hash** (a unique fingerprint) of its data, as well as the **hash of the previous block**.
* The hash is generated using a cryptographic algorithm like **SHA256**, which ensures that even a small change in the block’s data will produce a completely different hash (avalanche effect).

### **How Does This Prevent Forgery?**

* If a hacker tries to alter the data in a block, the hash of that block will change.
* Since each block contains the hash of the previous block, changing one block will **break the chain** because the next block’s "previous hash" will no longer match.
* To successfully forge the blockchain, the hacker would need to **recalculate the hashes of all subsequent blocks**, which is computationally infeasible.

## **2. Decentralization and Consensus Mechanisms**

### **What is Decentralization?**

* Blockchain operates on a **peer-to-peer (P2P) network**, where every participant (node) has a copy of the entire blockchain.
* There is no central authority controlling the network.

### **How Does This Prevent Forgery?**

* If a hacker tries to alter a block, they would need to **alter the same block on more than 50% of the nodes** in the network (this is known as a **51% attack**).
* Achieving this is extremely difficult and expensive because it requires controlling a majority of the network’s computational power.

## **3. Proof of Work (PoW) and Mining**

### **What is Proof of Work?**

* Miners compete to solve a **cryptographic puzzle** (finding a **nonce** that generates a hash below a target value).
* The first miner to solve the puzzle gets to add the block to the blockchain and is rewarded.

### **How Does This Prevent Forgery?**

* To forge a block, a hacker would need to **re-mine that block and all subsequent blocks**.
* This requires an enormous amount of computational power and time, making it practically impossible to alter the blockchain without being detected.

## **4. Network Consensus**

### **How Does the Network Agree on Valid Blocks?**

* When a miner successfully mines a block, it is broadcast to the entire network.
* Other nodes verify the block’s validity by checking:
  + The hash of the block.
  + The transactions inside the block.
  + The link to the previous block.
* If the block is valid, it is added to the blockchain, and all nodes update their copies.

### **What Happens if a Hacker Tries to Forge a Block?**

* If a hacker tries to introduce a forged block, the network will **reject it** because:
  + The block’s hash will not match the expected value.
  + The block will not be linked correctly to the previous block.
  + The transactions in the block may be invalid.

## **5. Real-World Example of Forgery Prevention**

### **Scenario:**

* A hacker tries to alter a transaction in **Block 3** of the blockchain.
* They change the transaction data and recalculate the hash of Block 3.
* However, Block 4 contains the hash of Block 3, which no longer matches the altered Block 3.
* The hacker must now recalculate the hash of Block 4, Block 5, and so on, which requires re-mining each block.

### **Why is This Hard?**

* Re-mining blocks requires solving the cryptographic puzzle for each block, which takes significant time and computational power.
* Meanwhile, the honest nodes in the network are continuously adding new blocks to the legitimate chain, making it even harder for the hacker to catch up.

## **6. Summary: How Blockchain Stops Forgery**

1. **Cryptographic Hashing**: Changing one block breaks the chain because the hashes no longer match.
2. **Decentralization**: A hacker would need to control more than 50% of the network to alter the blockchain.
3. **Proof of Work**: Re-mining blocks is computationally expensive and time-consuming.
4. **Network Consensus**: The network rejects invalid blocks and maintains the correct version of the blockchain.

Reflection

## **What is the "Reflection" in Blockchain?**

* The "reflection" is essentially the **correct and agreed-upon version of the blockchain** that the majority of nodes in the network have validated and accepted.
* After a designated time (or when a new block is added), all nodes compare their local copy of the blockchain with this **majority version** (the reflection).
* If a node’s copy doesn’t match the reflection, it means their copy is **outdated or incorrect**, so they replace it with the reflection to stay in sync with the network.

## **Why is This Important?**

* Blockchains are **decentralized**, meaning there’s no central authority to enforce the "correct" version of the blockchain.
* Nodes can sometimes have **different copies** due to network delays, errors, or malicious attacks.
* The reflection mechanism ensures that **all nodes eventually agree on the same version** of the blockchain, maintaining **consistency and trust** across the network.

## **How Does It Work?**

Here’s a step-by-step explanation of the process:

### **1. Majority Consensus**

* The blockchain network operates on a **consensus mechanism** (e.g., Proof of Work or Proof of Stake).
* When a new block is added, the majority of nodes validate it and agree that it’s correct.
* This **agreed-upon version** of the blockchain becomes the "reflection."

### **2. Periodic Synchronization**

* After a designated time (or when a new block is added), all nodes check their local copy of the blockchain against the reflection.
* If a node’s copy matches the reflection, it means they’re up to date.
* If a node’s copy **doesn’t match**, it means their copy is either:
  + **Outdated**: They missed some blocks due to network delays.
  + **Incorrect**: Their copy was tampered with or corrupted.

### **3. Updating the Local Copy**

* Nodes that don’t match the reflection **replace their local copy** with the reflection.
* This ensures that all nodes have the **same, correct version** of the blockchain.

## **What’s the Actual Concept Behind This?**

The "reflection" mechanism is based on the concept of **consensus** in blockchain networks. Here’s how it works in practice:

### **1. Consensus Mechanisms**

* Blockchains use consensus mechanisms like **Proof of Work (PoW)** or **Proof of Stake (PoS)** to ensure that all nodes agree on the state of the blockchain.
* For example, in Bitcoin (which uses PoW), miners compete to solve a cryptographic puzzle. The first miner to solve it gets to add a new block, and the network validates it.

### **2. Longest Chain Rule**

* In many blockchains (e.g., Bitcoin), the **longest chain** is considered the valid one. This is because the longest chain represents the version of the blockchain that has the most computational work (or stake) behind it.
* If a node has a shorter chain, it means they’re missing some blocks, so they update their chain to match the longest one.

### **3. Network Synchronization**

* Nodes constantly communicate with each other to share updates about the blockchain.
* If a node realizes its chain is shorter or different from the majority, it **replaces its chain** with the longer, agreed-upon version.

## **Why is This Mechanism Secure?**

1. **Decentralization**: No single node can control the reflection. It’s determined by the majority.
2. **Immutability**: Once a block is added to the reflection, it’s almost impossible to alter it without controlling the majority of the network.
3. **Consistency**: All nodes eventually have the same copy of the blockchain, ensuring trust and reliability.

5-6

## **1. Nonce and Mining**

### **What is a Nonce?**

* **Definition**: A **nonce** (number used once) is a 32-bit number that miners change to generate a valid hash for a block.
* **Purpose**: It’s used in the **Proof of Work (PoW)** process to solve the cryptographic puzzle and add a new block to the blockchain.
* **Range**: The nonce is a 32-bit unsigned number, meaning it can range from **0 to 4,294,967,296** (approximately 4 billion).

#### **Key Points:**

* Miners keep changing the nonce to find a hash that meets the network’s **target difficulty** (e.g., a hash with a certain number of leading zeros).
* The process of finding the correct nonce is called **mining**.

### **Golden Nonce**

* **Definition**: The **golden nonce** is the specific nonce value that produces a hash below the target difficulty.
* **Probability of Finding It**:
  + The probability of finding a valid hash (with 18 leading zeros) is extremely low: **0.000000000000000002%**.
  + Even with a 32-bit nonce range (4 billion possibilities), the probability of finding a valid hash is still very low: **0.000000001%**.
* **Conclusion**: One nonce range is **not enough** to guarantee finding the golden nonce, so miners often need to adjust other parameters (like the timestamp) and try again.

## **2. Timestamp in Blockchain**

### **What is a Timestamp?**

* **Definition**: A timestamp records the exact time and date when a block is mined and added to the blockchain.
* **Format**: It’s usually recorded in **Unix time** (the number of seconds since January 1, 1970).
* **Purpose**: Timestamps ensure that blocks are added in the correct chronological order.

#### **Example:**

* A block with the timestamp **1519181244** corresponds to **February 20, 2018, 10:47:24 UTC**.

## **3. Mining Process**

### **How Miners Pick Transactions**

* Miners select transactions from the **mempool** (a pool of unconfirmed transactions) to include in the next block.
* **Transaction Fees**: Miners prioritize transactions with higher fees because they earn these fees as rewards.
* **Block Configuration**: Miners adjust the block’s content (transactions, nonce, timestamp) to find a valid hash.

#### **Example:**

* In the mempool, transactions like:
  + **BAC1888**: Fee = 0.001 BTC
  + **AC700E5**: Fee = 0.0021 BTC
* Miners will prioritize transactions with higher fees (e.g., **AC700E5**) to maximize their earnings.

## **4. Mempool**

### **What is a Mempool?**

* **Definition**: The **mempool** (memory pool) is a temporary storage area for unconfirmed transactions waiting to be included in a block.
* **Function**: Miners select transactions from the mempool based on **fees** and other criteria.

#### **Key Points:**

* Transactions with higher fees are more likely to be picked by miners.
* If a transaction remains in the mempool for too long, it may be dropped or require a higher fee to be processed.

## **5. Consensus Protocols**

### **What is Consensus?**

* **Definition**: Consensus is the process by which nodes in a blockchain network agree on the validity of transactions and the state of the blockchain.
* **Purpose**: It ensures that all nodes have the **same copy** of the blockchain.

### **Types of Consensus Protocols:**

1. **Proof of Work (PoW)**:
   * Miners solve cryptographic puzzles to add blocks.
   * Used by Bitcoin and Ethereum (for now).
   * Energy-intensive but highly secure.
2. **Proof of Stake (PoS)**:
   * Validators are chosen based on the number of coins they hold and are willing to "stake" as collateral.
   * More energy-efficient than PoW.
   * Used by Ethereum 2.0 and other blockchains.
3. **Other Protocols**:
   * Delegated Proof of Stake (DPoS), Proof of Authority (PoA), etc.

## **6. Challenges in Blockchain**

### **Challenge 1: Attackers**

* **51% Attack**: If a single entity controls more than 50% of the network’s computational power (in PoW) or staked coins (in PoS), they can manipulate the blockchain.
* **Prevention**: Decentralization and consensus mechanisms make it extremely difficult and expensive to launch such attacks.

### **Challenge 2: Competing Chains**

* **Forks**: Sometimes, two miners solve the puzzle at the same time, creating two competing chains.
* **Resolution**: The network follows the **longest chain rule**, where the chain with the most work (or most blocks) is considered valid.

## **7. Block Validation Rules**

### **What Happens When a Block is Added?**

* Nodes in the network validate the new block using a set of rules:
  1. **Syntactic Correctness**: The block must be formatted correctly.
  2. **Non-Empty Transactions**: The block must contain at least one transaction.
  3. **Valid Hash**: The block’s hash must meet the target difficulty.
  4. **Timestamp Check**: The block’s timestamp must not be more than 2 hours in the future.
  5. **First Transaction**: The first transaction must be a **coinbase transaction** (reward for the miner).
  6. **Transaction Validation**: Each transaction in the block must be valid (e.g., correct signatures, no double-spending).
  7. **Block Reward**: The total block reward (coinbase + fees) must not exceed the maximum allowed.

## **8. Orphaned Blocks**

### **What are Orphaned Blocks?**

* **Definition**: Orphaned blocks are valid blocks that are not part of the main blockchain.
* **Cause**: They occur when two miners solve the puzzle at the same time, but only one chain becomes the main chain.
* **Resolution**: Orphaned blocks are discarded, and the transactions in them are returned to the mempool.

## **9. Key Terms to Remember**

* **Nonce**: A 32-bit number used in mining to find a valid hash.
* **Golden Nonce**: The nonce that produces a hash below the target difficulty.
* **Timestamp**: The time and date when a block is mined.
* **Mempool**: A pool of unconfirmed transactions waiting to be added to a block.
* **Consensus**: The process by which nodes agree on the state of the blockchain.
* **Proof of Work (PoW)**: A consensus mechanism where miners solve cryptographic puzzles.
* **Orphaned Blocks**: Valid blocks that are not part of the main chain.

✅ Latency affects which block gets accepted first.

✅ Majority decides the valid chain (longest chain rule).

✅ The orphaned block’s transactions are not lost but added back to the mempool.

✅ Proof-of-Work ensures that only one chain prevails.

9-10

## **Lecture 9: Blockchain Mining and Security Mechanisms**

### **Mempools: How Transactions Are Stored Before Mining**

* A **mempool** (memory pool) is where **unconfirmed transactions** are stored before they are included in a block.
* Each node maintains its own **mempool**, but in an ideal network, all nodes should eventually have the same transactions in their mempools.
* Mempools can differ temporarily due to network latency and local policies.
* All nodes eventually receive the same transactions and update their mempools to reflect the confirmed state of the blockchain.
* This ensures that the network maintains consistency and integrity over time.
* Transactions with **higher fees** are given priority since miners prefer transactions that reward them the most.

### **CPUs vs GPUs vs ASICs: Hardware Used in Mining**

| **Hardware** | **Processing Power** | **Energy Efficiency** | **Use Case** |
| --- | --- | --- | --- |
| **CPU (Central Processing Unit)** | Low | High Power Consumption | Not suitable for modern mining |
| **GPU (Graphics Processing Unit)** | Medium | Moderate | Used for Ethereum mining before PoS transition |
| **ASIC (Application-Specific Integrated Circuit)** | Very High | Low Power Consumption | Used for Bitcoin mining |

* **ASIC miners** are the most powerful and efficient but expensive.
* **GPU mining** is still used for certain cryptocurrencies, but Bitcoin mining is dominated by ASICs.

### **Mining Pools: Solving Blocks Collectively**

* A **mining pool** is a group of miners who share computational power to solve cryptographic puzzles faster.
* Rewards are distributed among participants based on their contribution to solving the block.
* Mining pools increase the chances of **regular rewards** compared to solo mining.

### **51% Attack: The Major Blockchain Threat**

* If a single miner or mining pool **controls more than 50%** of the network’s computing power, they can manipulate the blockchain.
* This would allow them to:
  + **Double spend** coins by reversing transactions.
  + **Prevent new transactions** from being added to the chain.
  + **Alter historical transactions**, rewriting blockchain history.
* **Prevention Measures:**
  + Decentralized mining by distributing hash power.
  + Using **Proof-of-Stake (PoS)** or hybrid consensus mechanisms.

### **Byzantine Fault Tolerance (BFT)**

* BFT ensures a blockchain remains secure even if some nodes behave maliciously.
* It states that a network can tolerate up to **1/3 of nodes being traitors** while still functioning correctly.
* **Key concept:** If honest nodes form a majority, the system remains secure and reaches consensus.

## **Lecture 10: Transactions and UTXO Model**

### **Understanding Transactions in Blockchain**

* A transaction in blockchain represents the transfer of assets (e.g., Bitcoin) from one address to another.
* Transactions are digitally signed using **private keys** to ensure authenticity.

### **UTXO: Unspent Transaction Output Model**

* **UTXO (Unspent Transaction Output)** represents the remaining balance after a transaction.
* Each transaction **consumes UTXOs** from previous transactions and creates **new UTXOs**.
* **Key Advantage:** Prevents **double spending** since each UTXO can only be spent once.

#### **Example 1: Basic UTXO Transaction**

1. Alice has **5 BTC** and wants to send **3 BTC** to Bob.
2. The transaction inputs:
   * **Alice's 5 BTC UTXO** is spent.
3. The transaction outputs:
   * **Bob receives 3 BTC** (new UTXO for Bob).
   * **Alice gets 2 BTC change** (new UTXO for Alice).

#### **Example 2: Complex UTXO Transaction**

1. Alice has two UTXOs: **2 BTC and 3 BTC**.
2. She wants to send **4 BTC** to Bob.
3. The transaction inputs:
   * Alice spends **both UTXOs (2 BTC + 3 BTC)**.
4. The transaction outputs:
   * Bob gets **4 BTC**.
   * Alice gets **1 BTC** in change.

### **Where Do Transaction Fees Come From?**

* **Transaction Fee = Inputs - Outputs**
* Miners collect transaction fees as an incentive to include transactions in a block.
* Users can **increase fees** to get their transactions confirmed faster.

### **UTXO of a Miner**

* Miners receive rewards in the form of **newly minted coins + transaction fees**.
* These rewards are also stored as UTXOs and can only be spent once they mature after a set number of blocks.

## **Summary of Key Learnings**

✅ **Mempools store pending transactions before they get added to a block.**✅ **ASIC miners dominate Bitcoin mining due to high efficiency.**✅ **Mining pools allow miners to combine resources and increase success rates.**✅ **A 51% attack can manipulate transactions, but decentralization prevents it.**✅ **Byzantine Fault Tolerance ensures the system remains secure even with some malicious nodes.**✅ **The UTXO model prevents double spending by ensuring each output can be used only once.**✅ **Transaction fees incentivize miners to prioritize high-fee transactions.**

9

**1. Mempools (Memory Pools)**

- Definition: A mempool is a temporary storage area in a blockchain node where unconfirmed transactions await inclusion in a block.

- Function:

1. Nodes broadcast transactions to the network.

2. Miners select transactions from the mempool to include in the next block.

3. Transactions are prioritized by fee-per-byte: Higher fees incentivize miners to pick them first.

**2. Mining Hardware: CPUs vs. GPUs vs. ASICs**

| Component | Role | Hash Rate | Use Case |
| --- | --- | --- | --- |
| CPU | General-purpose processing | < 10 MH/s | Low-efficiency mining (obsolete) |
| GPU | Parallel processing for mining | ~100–500 MH/s | Altcoins (e.g., Zcash, Ethereum) |
| ASIC | Custom-built for specific algorithms | > 1,000 GH/s | Bitcoin (SHA-256 algorithm) |

**- Key Insight:**

- ASICs dominate Bitcoin mining due to unmatched efficiency but lack flexibility.

- GPUs balance flexibility and power, making them popular for altcoins.

**- Economic Impact:**

- Slide 25 shows a GPU rig priced at $5,599, reflecting the capital-intensive nature of mining.

**3. Mining Pools**

- Definition: Collaborative groups where miners combine computational power to increase the probability of solving blocks.

**- Mechanics**:

1. Work is divided among participants (e.g., using the equation x = ½ to split tasks).

2. Rewards are distributed proportionally to contributed hash power.

**4. 51% Attack**

- Definition: An attack where a single entity controls >50% of the network’s hash rate, enabling double-spending and blockchain manipulation.

* **Potential Consequences:**
  + **Double Spending:** The attacker could reverse transactions that they made while in control, effectively double spending coins.
  + **Transaction Censorship:** The attacker could block or delay new transactions from being confirmed.
  + **Network Reorganization:** They could potentially disrupt the network’s consensus mechanism by reorganizing the blockchain.
* **Security Implications:** While executing a 51% attack is highly resource-intensive and costly, the risk underscores the importance of decentralization in maintaining blockchain security.

**- Probability & Cost:**

Attacking Bitcoin would require exorbitant resources (~$20 billion for ASICs), making it impractical.

**5. Byzantine Fault Tolerance (BFT)**

- Concept: A consensus protocol ensuring system reliability even if up to 1/3 of nodes are malicious ("traitors").

- Role in Blockchain:

- Used in permissioned blockchains (e.g., Hyperledger).

- "If ≤33% are traitors, the system remains secure."

- Comparison with PoW:

- BFT is energy-efficient but requires known, trusted validators.

**Sample Exam Questions**

**1. Q: Why are ASICs preferred for Bitcoin mining?**

A: ASICs offer unparalleled hash rates (>1,000 GH/s) for the SHA-256 algorithm, making them exponentially more efficient than CPUs/GPUs.

**2.Q: How does Byzantine Fault Tolerance (BFT) enhance blockchain security?**

A: Byzantine Fault Tolerance (BFT) enhances blockchain security through the following mechanisms:

| **Aspect** | **Explanation** |
| --- | --- |
| **Consensus Resilience** | BFT ensures that the blockchain network can achieve consensus even if up to 33% of nodes are malicious or faulty. This guarantees that honest nodes agree on the validity and order of transactions, preventing attacks like double-spending. |
| **Double-Spending Prevention** | By requiring a supermajority (e.g., 2/3) of nodes to validate transactions, BFT makes it impossible for malicious actors to alter the transaction history or approve fraudulent transactions. |
| **Energy Efficiency** | Unlike energy-intensive Proof of Work (PoW), BFT does not rely on computational puzzles. Instead, it uses voting or agreement protocols among nodes, drastically reducing energy consumption. |
| **Immediate Finality** | Transactions are finalized instantly once consensus is reached, eliminating the need for multiple confirmations (as in Bitcoin). This reduces vulnerability to short-term attacks. |
| **Use Case** | Ideal for permissioned blockchains (e.g., enterprise networks) where participants are known and trusted, ensuring fast and secure consensus. |

Trade-offs:

* Scalability: High communication overhead limits scalability in large, dynamic networks.
* Validator Set: Requires a fixed, known set of validators, making it less suitable for permissionless, public blockchains.

**Example**:  
In Hyperledger Fabric (a BFT-based blockchain), nodes validate transactions through a voting mechanism. If 4 out of 6 nodes approve, the transaction is added to the ledger. Even if 2 nodes are compromised, the network remains secure.

**3. Q: What determines transaction priority in a mempool?**

A: Fees. Miners prioritize transactions with higher fees to maximize rewards.

10

## **1.Transactions and UTXOs**

**Transactions** form the core of how cryptocurrencies are transferred between parties in a blockchain network. Unlike traditional bank transfers, blockchain transactions are public and use a model known as the UTXO (Unspent Transaction Output) model.

**UTXOs (Unspent Transaction Outputs)** represent the remaining value from previous transactions that can be used as inputs in new transactions. In essence, the UTXO model keeps track of discrete chunks of cryptocurrency, ensuring that each coin is spent only once.

## **2. Transactions: Structure and Process**

* **Definition:** A transaction is a data structure that specifies the transfer of value. It typically includes:
  + **Inputs:** References to previous UTXOs that the sender controls.
  + **Outputs:** New UTXOs created by the transaction, which specify the amount and the recipient.
* **Process:**
  + When a user wants to spend cryptocurrency, they reference one or more UTXOs they own as inputs.
  + The transaction then creates one or more outputs, which represent the new owners of the coins.
  + If the total input exceeds the desired output, the excess is sent back as “change” (another UTXO) to the sender.

## **3. UTXOs: Unspent Transaction Outputs**

* **Definition:** UTXOs are the leftover amounts from previous transactions that have not yet been spent.
* **Key Characteristics:**
  + **Discrete Units:** Each UTXO represents a specific amount of cryptocurrency.
  + **Immutability:** Once a UTXO is spent, it cannot be reused.
  + **Verification:** Nodes in the network verify that UTXOs used in a transaction have not been spent already, ensuring no double-spending occurs.
* **Comparison with Account Models:** Unlike account-based systems where balances are stored in a single account, the UTXO model treats coins as individual pieces that must be fully spent and then recreated as new outputs if necessary.

## **4. Detailed Examples of Transactions**

### **Example 1**

* **Scenario:** A simple transaction where a user sends a portion of a UTXO to another user.
* **Process:**
  + The user selects a UTXO worth, say, 1 BTC.
  + They decide to send 0.3 BTC to a recipient.
  + The transaction creates two outputs: one for the recipient (0.3 BTC) and one returning the change (0.7 BTC) to the sender.

### **Example 2**

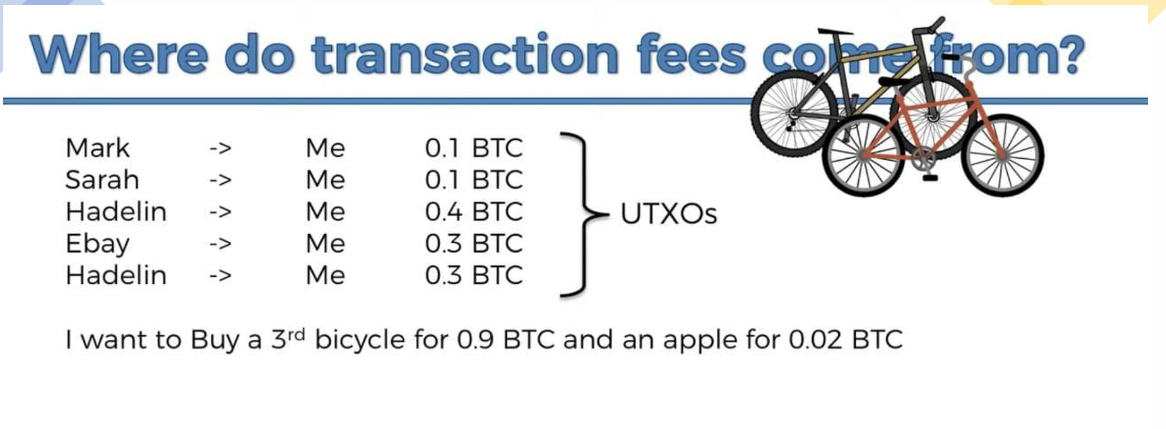
* **Scenario:** A more complex transaction with multiple inputs and outputs.
* **Process:**
  + A user may combine several UTXOs as inputs to reach a desired total amount.
  + The transaction then creates multiple outputs – for example, sending funds to multiple recipients and returning change.
* **Importance:** Such examples illustrate how the blockchain maintains a record of exactly which outputs remain unspent, thereby ensuring the integrity and traceability of every coin.

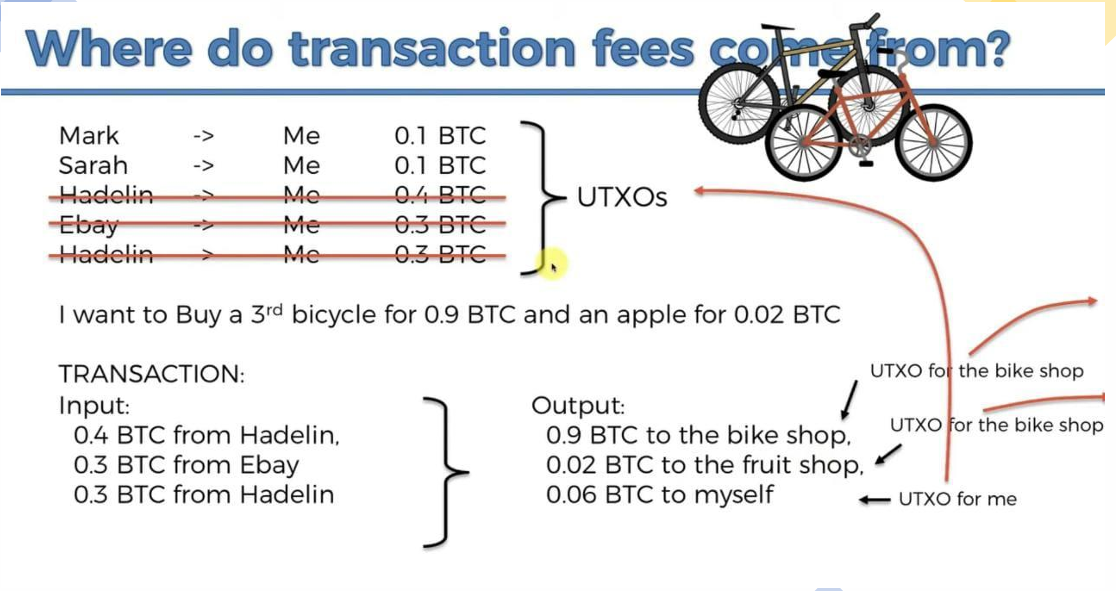
## **5. Transaction Fees**

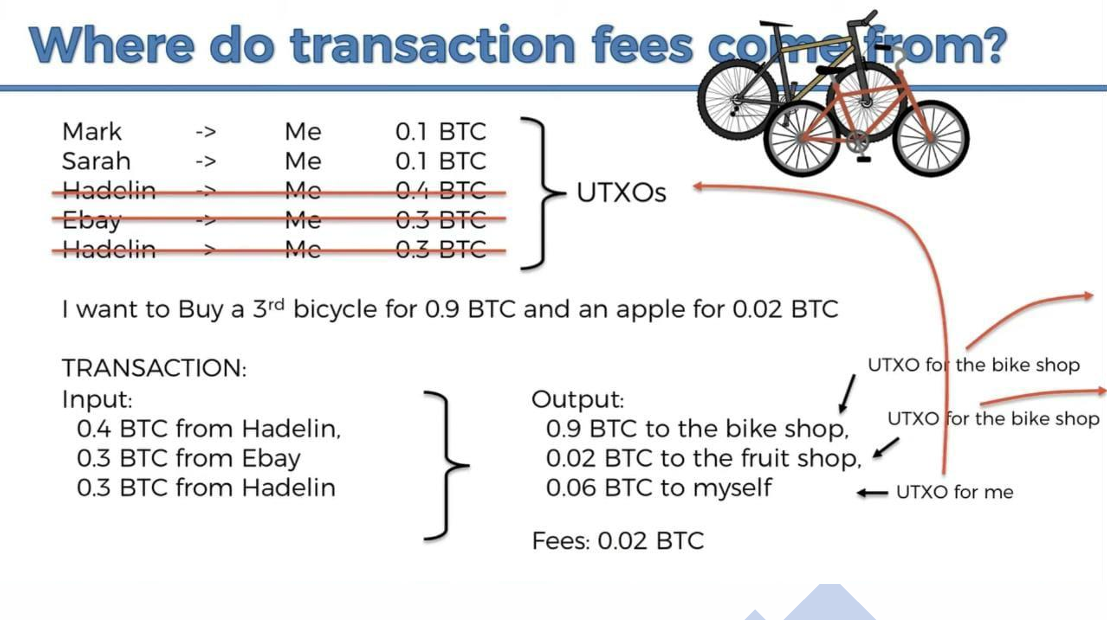
* **Purpose of Fees:**
  + **Incentive for Miners:** These fees reward miners for including the transaction in a block.
  + **Network Efficiency:** Fees also help prevent spam by incentivizing users to include appropriate fees.

## **6. UTXO of Miner (Coinbase Transaction)**

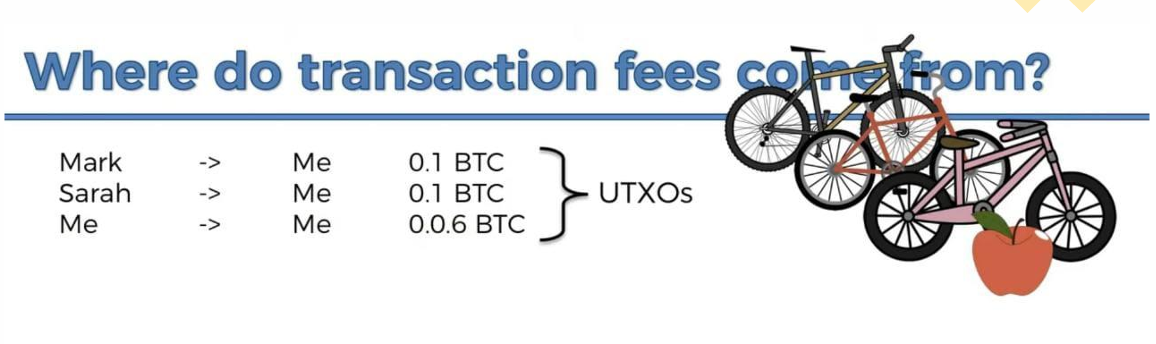
* **Definition:** The UTXO created by the miner in a newly mined block is known as the coinbase transaction.
* **Characteristics:**
  + **Block Reward:** This UTXO includes the block reward (newly minted coins) and the sum of transaction fees from all transactions included in that block.
  + **Special Handling:** Unlike regular transactions, the coinbase transaction has no inputs because it is the method by which new coins enter circulation.
* **Significance:** It is an essential mechanism that both rewards miners for their work and increases the circulating supply of the cryptocurrency in a controlled manner.

****

****

****

****

****

Golang by Murshad :(

We need to import stuff in golang to use it in the code. 🥺

# Imports:

**import (**

// Standard library imports

"fmt" // Provides formatted I/O operations (e.g., printing to console)

"log" // Used for logging events, errors, and debugging messages

"encoding/json" // Handles encoding and decoding of JSON data

"encoding/hex" // Provides functions for encoding and decoding hexadecimal strings

"crypto/sha256" // Implements the SHA-256 hashing algorithm for data integrity and security

"crypto/ecdsa" // Implements the Elliptic Curve Digital Signature Algorithm for signing data

"crypto/elliptic" // Provides elliptic curve cryptography operations used alongside ECDSA

"crypto/rand" // Generates cryptographically secure random numbers

"math/big" // Supports arbitrary-precision arithmetic,for large numbers in cryptographic calculations

"time" // Manages time-related functions, such as timestamps and timeouts

**)**

# Variable Declaration

**Declaring Variables**

var name string = "Alice" // Explicit type declaration

age := 30 // Type inferred (int)

balance := 100.50 // Type inferred (float64)

const age = 30 // Cannot be changed once declared/initialized

# Loopy

***Go has only for loops***

// Basic loop

for i := 0; i < 5; i++ {

fmt.Println(i)

}

// While-like loop

n := 0

for n < 5 {

fmt.Println(n)

n++

}

# Maps

// Initialize a map to track balances

balances := make(map[string]int)

balances["Alice"] = 100

balances["Bob"] = 50

// Check if a key exists

if bal, exists := balances["Charlie"]; exists {

fmt.Println("Charlie's balance:", bal)

} else {

fmt.Println("Charlie not found")

}

// Delete a key

delete(balances, "Bob")

// Loop over a map

for key, value := range balances {

fmt.Printf("%s: %d\n", key, value)

}

# Pointers

func updateBalance(user \*User, newBalance int) {

user.Balance = newBalance // Dereference automatically with dot notation

}

func main() {

user := &User{Name: "Alice", Balance: 100} // & gets the memory address

updateBalance(user, 200)

fmt.Println(user.Balance) // Output: 200

}

# Slices (Dynamic Arrays)

// Initialize a slice

numbers := [ ]int{1, 2, 3}

// Append elements

numbers = append(numbers, 4, 5)

// Slice a slice

subset := numbers[1:3] // →[2, 3]

Murshad itni mehnat na kro yar

Ly ly ly hai

Hahaha ali bhai ly ly ly

# SAMPLE CODE:

package main

import (

"errors"

"fmt"

)

// 1. Define a struct for a Book

type Book struct {

Title string

Author string

ISBN string

}

// 2. Define a struct for the Library (analogous to the blockchain)

type Library struct {

Books []\*Book // Slice of pointers to Book (like blocks in a chain)

}

// 3. Add a book to the library (similar to InsertBlock)

func (lib \*Library) AddBook(title, author, isbn string) {

newBook := &Book{

Title: title,

Author: author,

ISBN: isbn,

}

lib.Books = append(lib.Books, newBook)

}

// 4. Remove a book by ISBN (similar to changing a block)

func (lib \*Library) RemoveBook(isbn string) error {

for i, book := range lib.Books {

if book.ISBN == isbn {

// Remove the book by slicing

lib.Books = append(lib.Books[:i], lib.Books[i+1:]...)

return nil

}

}

return errors.New("book not found")

}

// 5. List all books (similar to ListBlocks)

func (lib \*Library) ListBooks() {

fmt.Println("Library Catalog:")

for idx, book := range lib.Books {

fmt.Printf("Book %d:\n", idx+1)

fmt.Println(" Title:", book.Title)

fmt.Println(" Author:", book.Author)

fmt.Println(" ISBN:", book.ISBN)

fmt.Println("--------------------------")

}

}

// 6. Find a book by ISBN (similar to searching for a transaction)

func (lib \*Library) FindBook(isbn string) (\*Book, error) {

for \_, book := range lib.Books {

if book.ISBN == isbn {

return book, nil

}

}

return nil, errors.New("book not found")

}

func main() {

// Initialize an empty library

library := &Library{}

// Add books to the library

library.AddBook("The Go Programming Language", "Alan A. A. Donovan", "978-0134190440")

library.AddBook("Clean Code", "Robert C. Martin", "978-0132350884")

// List all books

library.ListBooks()

// Try to remove a book

err := library.RemoveBook("978-0134190440")

if err != nil {

fmt.Println("Error:", err)

} else {

fmt.Println("Book removed successfully!")

}

// List updated catalog

library.ListBooks()

// Search for a book

book, err := library.FindBook("978-0132350884")

if err != nil {

fmt.Println("Error:", err)

} else {

fmt.Println("Found book:", book.Title)

}

}

### COMMENTS FOR ABOVE CODE:

**1. Structs (Book and Library)**

* **Purpose**: Define custom data types (like Block in the blockchain).
* **Syntax**:

type Book struct {

Title string

Author string

}

* **Analogy**: Book is like a single block, and Library is the chain holding slices of books.

#### **2. Pointers**

* **Usage**: lib \*Library in methods and []\*Book (slice of pointers).
* **Why?**
  + Avoid copying large structs (like blocks in a blockchain).
  + Modify the original data (e.g., when removing a book).

#### **3. Slices ([]\*Book)**

* **Dynamic Arrays**: Used to store a growing list of books (like transactions in a block).
* **Operations**:
  + Append: lib.Books = append(lib.Books, newBook)
  + Remove: lib.Books = append(lib.Books[:i], lib.Books[i+1:]...)

#### **4. Methods (AddBook, RemoveBook)**

* **Receiver Syntax**: func (lib \*Library) AddBook(...).
* **Pointer Receivers**: Used to modify the library’s state (like InsertBlock modifies the chain).

#### **5. Error Handling**

* **Example**:

func (lib \*Library) RemoveBook(isbn string) error {

// ...

return errors.New("book not found")

}

* **Analogy**: Similar to ChangeBlock checking if a transaction exists.

#### **6. Loops (for with range)**

* **Traversing Slices**:

for idx, book := range lib.Books { ... }

* **Analogy**: Like traversing blocks in ListBlocks or VerifyChain.

GoLang by Umamah :D

# Syntax for Key Concepts

#### **1. Structs (Defining a Block)**

// Block represents a block in the blockchain

type Block struct

{

transactions [ ] string

prevPointer \*Block

prevHash string

currentHash string

}

#### **2. Pointers (Linking Blocks)**

var chainHead \*Block // Points to the latest block

#### **3. Maps (Storing Transaction Data)**

Spender := make(map[string]int)

Receiver := make(map[string]int)

#### **4. Slices (Handling Multiple Transactions)**

transactions := []string{"Tx1", "Tx2"}

#### **5. Functions (Example: Calculate Balance)**

func CalculateBalance ( userName string, chainHead \*Block ) int

{

balance := 0

tempBlock := chainHead

for tempBlock != nil

{

balance += tempBlock.Receiver[userName] - tempBlock.Spender[userName]

tempBlock = tempBlock.prevPointer

}

return balance

}

#### **6. Loops (Iterating Over Blockchain)**

for tempBlock != nil

{

fmt.Println(tempBlock.transactions)

tempBlock = tempBlock.prevPointer

}

#### **7. Hashing (SHA-256)**

import (

"crypto/sha256"

"encoding/hex"

"fmt"

)

// CalculateHash computes the hash of a block: prevHash + data

func CalculateHash ( block \*Block ) string {

data := block.prevHash

for \_ , transaction := range block.transactions {

data += transaction

}

hash := sha256.Sum256( [ ] byte ( data ) )

return hex.EncodeToString( hash [ : ] ) //convert the array of 32 bytes into a string

}

#### **8. Conditionals (Blockchain Verification)**

// Check if the calculated hash matches the stored hash

if calculatedHash != currentBlock.currentHash {

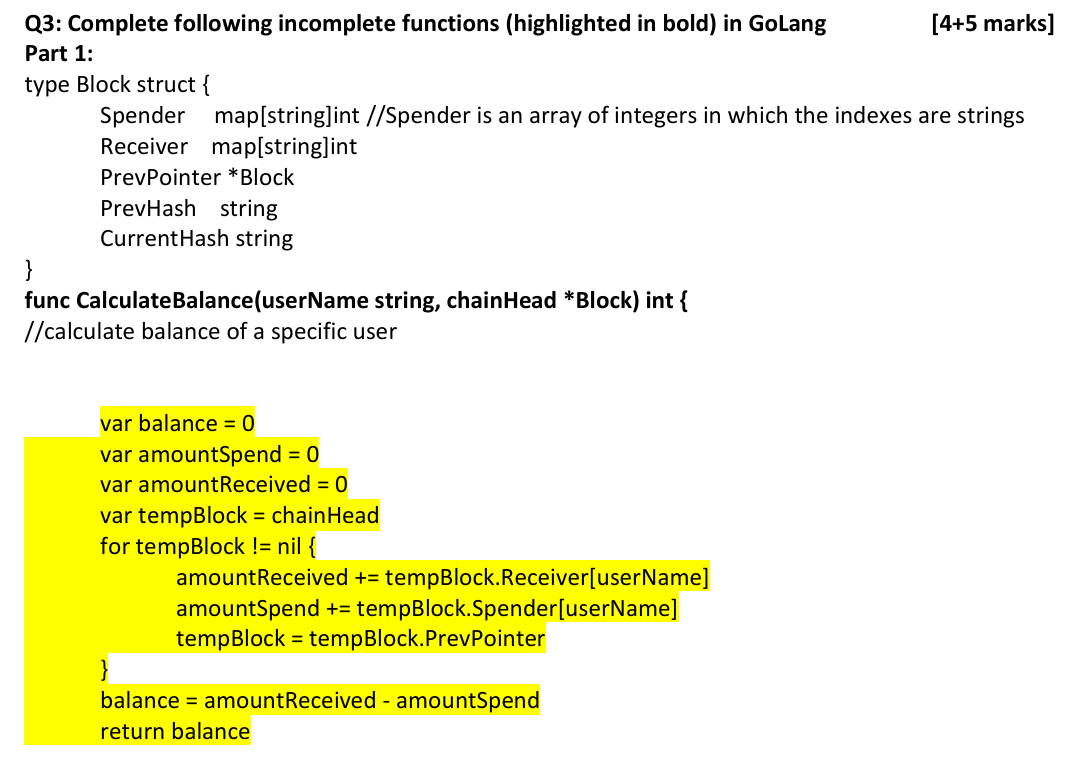
fmt.Println("Blockchain is compromised! Hash mismatch in block.")

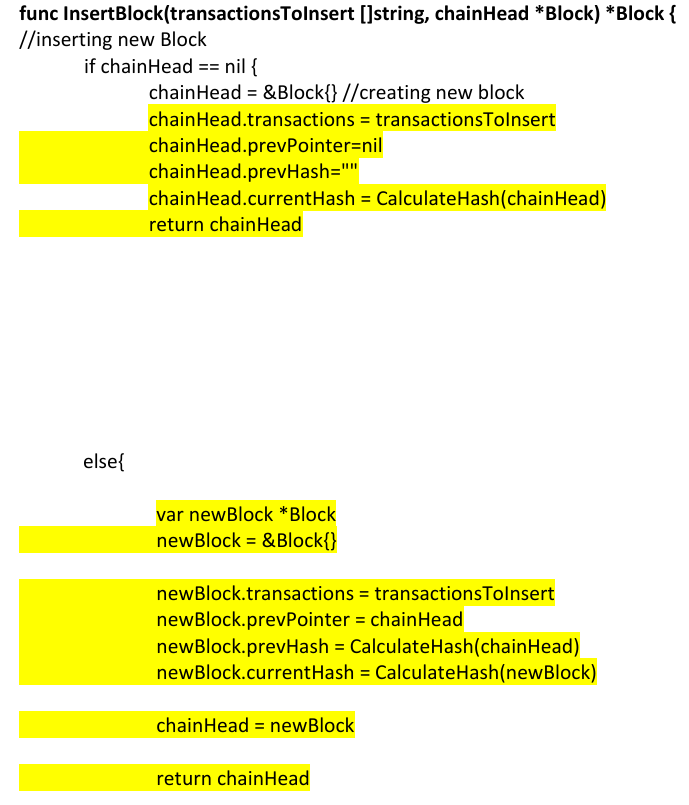
return

}

#### **9. Printing & Debugging**

fmt.Println("Blockchain is valid!")





// VerifyChain checks if the blockchain is valid

func VerifyChain(chainHead \*Block) {

// Start from the head of the chain

currentBlock := chainHead

// Traverse the blockchain

for currentBlock != nil {

// Recalculate the hash of the current block

calculatedHash := CalculateHash(currentBlock)

// Check if the calculated hash matches the stored hash

if calculatedHash != currentBlock.currentHash {

fmt.Println("Blockchain is compromised! Hash mismatch in block.")

return

}

// Check if the previous block exists

if currentBlock.prevPointer != nil {

// Verify that the prevHash of the current block matches the currentHash of the previous block

if currentBlock.prevHash != currentBlock.prevPointer.currentHash {

fmt.Println("Blockchain is compromised! Previous hash mismatch.")

return

}

}

// Move to the previous block

currentBlock = currentBlock.prevPointer

}

// If all blocks are valid

fmt.Println("Blockchain is unchanged and valid.")

}

// CalculateHash computes the hash of a block: prevHash + data

func CalculateHash(block \*Block) string {

data := block.prevHash

for \_, transaction := range block.transactions {

data += transaction

}

hash := sha256.Sum256([]byte(data))

return hex.EncodeToString(hash[:]) //convert the array of 32 bytes into a string

}

// ChangeBlock modifies a transaction in a block

func ChangeBlock(oldTrans string, newTrans string, chainHead \*Block) {

currentBlock := chainHead

for currentBlock != nil {

for i, transaction := range currentBlock.transactions {

if transaction == oldTrans {

currentBlock.transactions[i] = newTrans

currentBlock.currentHash = CalculateHash(currentBlock)

fmt.Println("Transaction changed successfully!")

return

}

}

currentBlock = currentBlock.prevPointer

}

fmt.Println("Transaction not found in the blockchain.")

}

// ListBlocks displays all blocks in the blockchain

func ListBlocks(chainHead \*Block) {

currentBlock := chainHead

blockNumber := 1

for currentBlock != nil {

fmt.Printf("Block %d:\n", blockNumber)

fmt.Println("Transactions:", currentBlock.transactions)

fmt.Println("Previous Hash:", currentBlock.prevHash)

fmt.Println("Current Hash:", currentBlock.currentHash)

fmt.Println("-------------------------")

currentBlock = currentBlock.prevPointer

blockNumber++

}

}

Class Activity

### **1. Introduction**

* Existing online payments rely on financial institutions, which act as intermediaries.
* This trust-based model leads to costs, transaction reversals, and fraud.
* A **peer-to-peer (P2P) electronic cash system** can solve these issues.
* The proposed system eliminates the need for a trusted third party by using cryptographic proof.

### **2. Transactions**

* Bitcoin transactions use a **chain of digital signatures**.
* Each owner transfers coins by signing the previous transaction and the recipient’s public key.
* The challenge is preventing **double-spending** without a central authority.

### **3. Timestamp Server**

* Transactions are grouped into blocks and **timestamped**.
* Each block references the previous one, creating an immutable chain.
* This forms a chronological record of transactions.

### **4. Proof-of-Work (PoW)**

* Uses **computational puzzles** (like Hashcash) to secure the blockchain.
* Miners must find a hash with a required number of leading zeros.
* The longest chain represents the most CPU effort and is accepted as the valid chain.
* **Modifying past transactions requires redoing all the work, making attacks impractical**.

### **5. Network Protocol**

1. Transactions are broadcasted to all nodes.
2. Nodes collect transactions into blocks.
3. Miners compete to find a valid proof-of-work.
4. The new block is broadcasted and verified.
5. Nodes accept the longest valid chain.

### **6. Incentives**

* Miners receive rewards in the form of **newly created bitcoins and transaction fees**.
* This incentivizes network security and prevents malicious activity.

### **7. Disk Space Optimization**

* Transactions are stored in a **Merkle Tree**, allowing old transactions to be pruned.
* Only block headers (80 bytes each) need to be kept, reducing storage requirements.

### **8. Simplified Payment Verification (SPV)**

* Users can verify transactions without running a full node.
* SPV relies on the **longest blockchain** and Merkle proofs.

### **9. Combining and Splitting Transactions**

* Bitcoin allows transactions with **multiple inputs and outputs**.
* This enables efficient transfers and handling of change.

### **10. Privacy**

* Unlike banks, Bitcoin does not store identities, only public keys.
* Privacy is maintained as **transactions are pseudonymous**.
* However, multi-input transactions can reveal ownership links.

### **11. Security and Attack Analysis**

* An attacker trying to rewrite history must outpace the honest chain.
* The probability of success decreases **exponentially** as more blocks are added.
* This makes Bitcoin secure as long as the majority of computing power is honest.

### **12. Conclusion**

* Bitcoin provides a decentralized, trustless system for digital payments.
* Proof-of-work ensures security and prevents fraud.
* Consensus is achieved by **CPU-based voting**, making the system resilient.

13,14

**Three Layers in Crypto:**

**Technology**: Underlying infrastructure (e.g., blockchain).

**Protocol/Coin**: Rules governing the network (e.g., Bitcoin Protocol).



**Token**: Digital assets (e.g., BTC, ETH).

**Bitcoin Protocol**

The Bitcoin protocol is the foundational set of rules that governs how participants in the Bitcoin network communicate and reach agreement. It covers:

* How network participants **communicate** with each other
* **Consensus mechanisms** for transaction validation
* The use of public keys and digital signatures for **authentication**
* **Governance processes** for protocol updates

Think of the protocol as the constitution of the Bitcoin network—it defines all the rules that make the system work as intended. Just as HTTP is a protocol that enables web communication, the Bitcoin protocol enables decentralized financial transactions. All in all, a set of rules dictating how nodes communicate, how transactions are validated (using digital signatures based on public-private keys), and how consensus is achieved.

**Monetary Policy:**

Bitcoin’s monetary policy is coded into its protocol. It is predetermined. It includes:

* A fixed maximum supply (21 million bitcoins).
* A block reward halving mechanism(halve the reward after every 210,000 blocks are added to the blockchain)—initially 50 BTC per block, then 25, 12.5, 6.25 (and further down)—to control issuance and mimic scarcity.
* Block rewards that halve approximately every four years (known as "halving")
* Gradually decreasing inflation rate, eventually reaching zero
* No central authority can change these monetary rules
* Block Frequency is10 minutes per block (Bitcoin) vs. 15 seconds (Ethereum).

This fixed monetary policy makes Bitcoin fundamentally different from fiat currencies, which can be printed at will by central banks. The predictable issuance schedule creates digital scarcity and is often compared to digital gold.

**Consensus Algorithm: Proof of Work**

Bitcoin uses Proof of Work (PoW) as its consensus mechanism. Here's how it works:

1. Miners compete to solve complex mathematical puzzles that require significant computational power
2. The first miner to solve the puzzle gets to add a new block to the blockchain
3. The solution must be difficult to find but easy to verify by other network participants
4. This process secures the network against attacks by making it prohibitively expensive to alter past transactions

The work itself involves finding a hash value that meets certain criteria (typically beginning with a specific number of zeros). This requires miners to repeatedly try different nonce values until they find a valid solution.

**Mining Incentives**

Miners are incentivized to participate in the network through:

1. Block rewards: New bitcoins created and awarded to the miner who successfully mines a block
2. Transaction fees: Fees paid by users to have their transactions included in blocks

The current block reward plus transaction fees provide financial motivation for miners to contribute computing power to secure the network. As block rewards diminish over time according to the halving schedule, transaction fees are expected to become the primary incentive.

**Public and Private Keys, Digital Signatures**

Bitcoin uses asymmetric cryptography with key pairs:

* Private key: A secret number known only to the owner, used to sign transactions
* Public key: Derived from the private key, can be shared with anyone
* Bitcoin address: A hashed version of the public key, used as the "account number" to receive funds

When you "send" bitcoin, you're actually using your private key to create a digital signature proving you have the right to spend from a particular address. This signature can be verified by anyone using your public key, but the private key remains secret.

**HD (Hierarchically Deterministic) Wallets**

HD wallets improve security and usability by:

* Generating multiple private-public key pairs from a single master seed
* Creating a deterministic sequence of keys that can be recreated using the seed phrase
* Allowing users to backup all their keys with a single seed phrase (typically 12 or 24 words)
* Providing improved privacy by using a new address for each transaction

This hierarchical structure means users can recover all their keys and addresses by remembering just one seed phrase, rather than backing up each private key individually.

19

**Smart Contracts**

Smart contracts are self-executing pieces of code stored on a blockchain. They automatically enforce and execute agreements when predetermined conditions are met. Key characteristics include:

* Code that automatically executes predefined actions when certain conditions are met
* Immutable once deployed (cannot be changed)
* Transparent and verifiable by all network participants
* Trustless execution without intermediaries

**Ethereum Virtual Machine (EVM):** Executes smart contracts in a sandboxed environment.

**Key Use-Cases:**

They underpin decentralized applications (DApps) and can govern anything from simple transactions to complex financial instruments.

**Ethereum Virtual Machine (EVM)**

The EVM is the runtime environment where all Ethereum smart contracts execute:

* It's a sandboxed, isolated environment for security purposes
* Prevents smart contracts from accessing the host computer's file system
* Provides a standardized execution environment across all Ethereum nodes
* Enables deterministic execution (same input always produces same output)

**Security Considerations:**

**Vulnerabilities**:

* Risks include viruses, unauthorized access to private data
* Infinite loops, or heavy computations that could stall the network.

**Mitigation**:

* The Ethereum Virtual Machine (EVM) helps manage security risks by isolating the execution of smart contracts
* Using “gas” as a cost measure for every computation to prevent infinite loops or excessively heavy computations.

**Gas and Why It Matters**

Gas is a unit that measures the amount of computational effort required to execute operations (such as running a smart contract). Every operation on the blockchain(ethereum) has a gas cost.

**Gas System**

Ethereum's gas system solves two critical problems:

1. **Security**: Prevents infinite loops and resource abuse by requiring payment for computation. Developers are penalized (by paying more) for inefficient or “heavy” code.
2. **Network Protection**: Prevents misuse such as infinite loops by imposing a cost for every computation.
3. **Resource allocation**: Creates a market mechanism for allocating network resources

Gas works as follows:

* Each operation in the EVM costs a predefined amount of gas
* Developers/users must pay for this gas using ETH
* Gas price fluctuates based on network demand
* Transactions specify a gas limit (maximum gas willing to use)

If a contract runs out of gas during execution, the operation reverts, but the gas is still consumed. This incentivizes efficient code writing and prevents network abuse.

**Gas vs. Ether:**

Although gas is paid in Ether (the native token of Ethereum), the gas system separates the measure of computational effort from the currency itself.

Gas cost is ether’s 1 billionth.

**Why use gas cost, why not ether?**

The separation of gas and Ether in Ethereum's economic model is actually a brilliant design decision that solves several important problems. Let me explain why Ethereum uses gas instead of pricing operations directly in Ether.

**Separation of Computational Cost and Market Value**

Gas represents the fixed computational cost of operations on the Ethereum Virtual Machine (EVM). Each operation has a predetermined gas cost based on the computational resources it requires. For example:

* Adding two numbers costs 3 gas
* Storing a value in contract storage costs 20,000 gas
* Reading from storage costs 200 gas

These costs remain constant regardless of Ether's market price. Even if gas cost fluctuates it is minimal (usually practically negligible). If operations were priced directly in Ether, the computational costs would fluctuate wildly with Ether's market value. Imagine if storing a value cost 0.001 ETH directly:

* When ETH is worth $100, that's $0.10
* When ETH rises to $4,000, suddenly the same operation costs $4.00!

This would make the network practically unusable during price surges, as simple operations would become prohibitively expensive.

**Market-Based Resource Allocation**

The gas system creates a two-part pricing model:

1. **Gas cost**: Fixed amount of gas required for an operation (e.g., 20,000 gas to store a value)
2. **Gas price**: How much Ether you're willing to pay per unit of gas (e.g., 50 Gwei per gas unit)

This separation allows users to express how urgently they need their transaction processed. During network congestion, users who need fast processing can offer a higher gas price, while those who can wait can offer less. This creates an efficient market for block space without changing the fundamental computational costs.

**Adaptability to Network Conditions**

The separation allows the protocol to adjust computational costs (gas costs) independently from the market value of transactions (gas price):

* EIP-1559 introduced a base fee that adjusts automatically based on network congestion
* Various Ethereum upgrades have adjusted the gas costs of specific operations to better reflect their true computational cost
* Users can prioritize their transactions based on urgency without changing the fundamental cost accounting

**Standardization Across Diverse Operations**

Different operations in Ethereum consume varying amounts of computational power. Gas standardizes these costs by assigning a gas cost to every possible operation within the Ethereum Virtual Machine (EVM). This makes it easier for developers to understand how much their code will cost to run, independent of fluctuations in Ether’s price.

An Illustrative Analogy

Think of gas like the electricity required to run appliances in your home:

1. **Gas cost** = kilowatt-hours required to run different appliances  
   * A toaster might need 0.04 kWh
   * A washing machine might need 0.5 kWh
2. **Gas price** = the rate you pay per kilowatt-hour  
   * During peak hours, electricity might cost $0.30/kWh
   * During off-peak hours, it might cost $0.10/kWh
3. **Gas limit** = the capacity of your home's circuit breaker (prevents electrical fires)
4. **Total cost** = kWh × rate

This system ensures you're charged based on your actual electricity usage, can choose to run your appliances at different times based on urgency and price, and has a safety mechanism to prevent catastrophic failure.

Similarly, Ethereum's gas system ensures users pay for their actual computational usage, can prioritize based on urgency, and includes safety mechanisms to protect the network.

**Decentralized Applications (DApps) and DAOs**

**DApps:**

Decentralized Applications are applications that run on a blockchain network (like Ethereum) rather than centralized servers

DApps are applications built on blockchain platforms that have:

* Backend code (smart contracts) running on a decentralized network
* No central point of control or failure
* Transparent and verifiable operation
* Often an interface built with standard web technologies

**DAOs (Decentralized Autonomous Organizations):**

Organizations or collectives that operate through smart contracts—autonomously enforcing rules without centralized management.

DAOs represent organizations governed by code rather than people:

* Rules encoded in smart contracts
* Governance decisions made through token-holder voting
* Treasury managed autonomously according to codified rules
* Operations executed automatically without human intermediaries

DAOs can be thought of as "organizations without managers" where stakeholders collectively make decisions through voting mechanisms, and the resulting actions are automatically executed by smart contracts.

**SAMPLE QUESTIONS**

### **Question 3: Ethereum Gas System**

**Q: Explain Ethereum's gas system and why it's necessary for the network.**

**A:** Ethereum's gas system is a mechanism for allocating computational resources and preventing abuse on the network. Gas is a unit of measurement for computational effort, with each operation in the Ethereum Virtual Machine (EVM) assigned a specific gas cost.

The system is necessary for several reasons:

1. **Preventing infinite loops**: By requiring payment for computation, it prevents malicious or accidental infinite loops from consuming network resources.
2. **Resource allocation**: It creates a market-based approach to allocating limited network resources.
3. **Compensating validators**: Gas fees provide incentives for validators to process and verify transactions.
4. **Network economics**: The gas system creates a separation between ETH's price and the cost of using the network.

When users submit transactions, they specify a gas limit (maximum gas they're willing to use) and a gas price (how much they're willing to pay per unit of gas). If a transaction runs out of gas during execution, it reverts, but gas used is still paid to prevent denial-of-service attacks.

### **Question 4: Proof of Work**

**Q: Describe Bitcoin's Proof of Work consensus mechanism and explain its security implications.**

**A:** Bitcoin's Proof of Work (PoW) is a consensus mechanism where miners compete to solve computational puzzles to validate transactions and create new blocks. The process works as follows:

1. Miners collect pending transactions into a block
2. They repeatedly modify a nonce value and calculate the block's hash
3. They search for a hash that meets a specific difficulty target (starts with a certain number of zeros)
4. The first miner to find a valid hash broadcasts the block to the network
5. Other nodes verify the solution is correct and add the block to their copy of the blockchain

Security implications:

* **51% Attack Resistance**: An attacker would need to control more than 50% of the network's total computing power to mount a successful attack
* **Immutability**: Altering past transactions requires redoing the PoW for that block and all subsequent blocks
* **Economic Security**: The cost of attacking the network exceeds the potential benefits
* **Decentralization**: Anyone with computing hardware can participate in mining, preventing centralized control

However, PoW also has drawbacks including high energy consumption and the tendency toward mining centralization due to economies of scale.

### **Question 5: HD Wallets**

**Q: What are Hierarchical Deterministic (HD) wallets and what advantages do they offer over traditional wallets?**

**A:** Hierarchical Deterministic (HD) wallets are cryptocurrency wallets that generate a structured tree of private/public key pairs from a single master seed. Their advantages include:

1. **Simplified backup**: Users only need to back up the master seed (typically as a 12-24 word mnemonic phrase) rather than individual private keys
2. **Enhanced privacy**: They can generate a new address for each transaction without requiring additional backups
3. **Structured organization**: Keys can be organized hierarchically for different purposes (e.g., separate branches for different cryptocurrencies or accounts)
4. **Offline key generation**: New addresses can be created without exposing the private keys to a networked device
5. **Account recovery**: The entire wallet with all addresses and funds can be reconstructed from the seed phrase on a new device

This is significantly more convenient and secure than traditional wallets where each private key needs to be individually backed up, creating both security risks and management challenges.

20

Bitcoin Consensus Algorithm & Proof-of-Work (PoW)

* **What is PoW?** PoW is a consensus mechanism where miners expend computational resources (i.e., perform “work”) to solve a cryptographic puzzle (hash puzzle). This process is competitive and secures the network.
* **Key Properties of PoW:**
* **Difficult to Compute:** The puzzle is designed to be computationally intensive so that finding a solution requires many attempts. The puzzle is intentionally hard to solve. This means miners must try many different numbers (nonces) by trial and error until they find one that makes the block’s hash meet the target.It takes a lot of computer power and time, which helps secure the network against attacks.
* **Parameterizable Cost:** The network adjusts the difficulty approximately every two weeks to keep block creation at an average of 10 minutes, regardless of hardware improvements. The point is to roughly maintain the duration of addition of blocks. In other words, the “cost” (in computational effort) to solve the puzzle can be set higher or lower by changing the target—keeping the block-production time stable.
* **Trivial to Verify:** Once a solution is found, verifying that the hash meets the target is quick and easy. All other nodes only need to recalculate one hash of the block’s data to confirm that the solution is correct—this small check takes very little time or power compared to finding the solution.
* **Incentives for Miners:**
* **Block Reward:** Miners receive a fixed amount of newly minted bitcoins when they add a valid block.
* **Transaction Fees:** In addition to the block reward, miners collect fees from the transactions included in the block.

**Wallets and Cryptography**

**What is a Cryptocurrency Wallet?**

**Definition:** A cryptocurrency wallet is a software (or hardware) tool that stores the cryptographic keys—specifically the private and public keys—needed to access and manage your digital assets (such as Bitcoin or Ether).

**Primary Functions:**

* **Storage of Keys:** Securely maintains your private (secret) and public keys.
* **Transaction Management:** Allows you to create, sign, and send transactions on the blockchain.
* **Balance Monitoring:** Provides an interface to view your holdings and track transactions.

**Public & Private Keys:**

* **Private Key:** Secret number used to sign transactions—if lost, funds may be unrecoverable.
* **Public Key:** Derived from the private key and shared with the network to receive funds.

**Bitcoin Address:**

A shorter representation (often via a hash of the public key) that others use to send bitcoins.A user’s "public-facing" identifier that is derived from their public key, typically via a hashing process (SHA-256 followed by RIPEMD-160) and some encoding (e.g., Base58Check).

**Purpose of Hashing:**Hash functions provide security and privacy by creating a fixed-length output that masks the public key, making it impractical for an attacker to reverse-engineer the original key.

**Hierarchical Deterministic (HD) Wallets:**Wallets that can generate multiple private-public key pairs from one seed, improving security and privacy by enabling address reuse avoidance. (A type of wallet that generates a tree of key pairs from a single seed phrase (mnemonic).)

**Benefits:**

* Simplified Backup: One seed phrase can recover all generated keys.
* Enhanced Privacy: New addresses can be generated for every transaction, reducing the possibility of linking transactions to one identity.

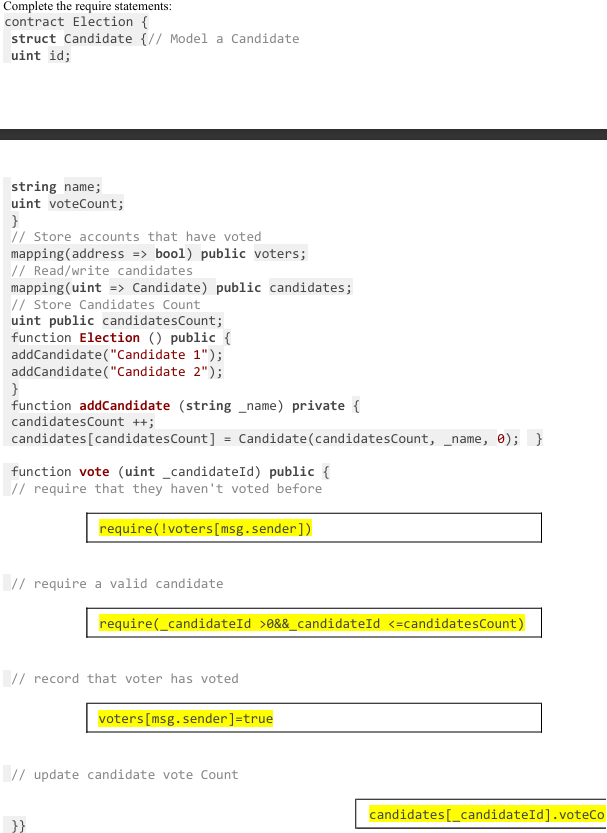
**SAMPLE QUESTIONS**

**Discuss the importance of cryptographic hash functions in the context of wallet security.**

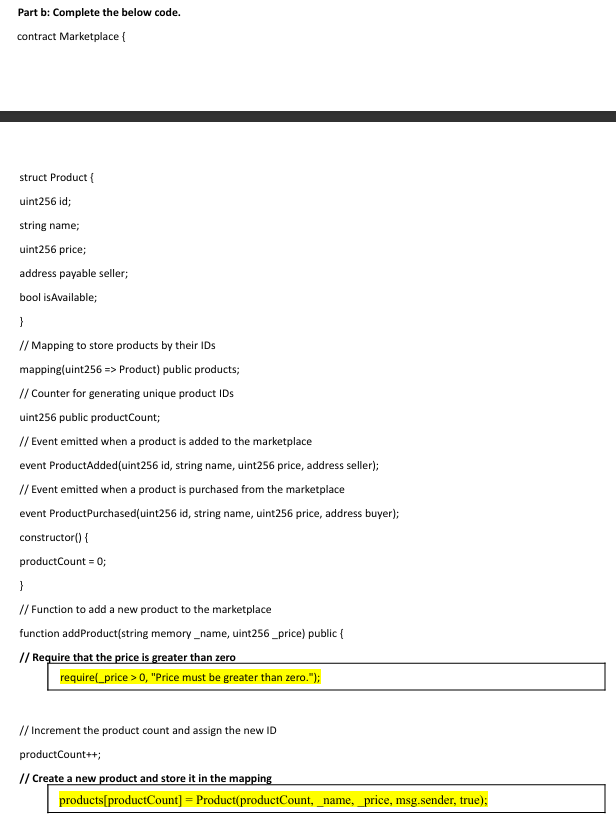
**Answer:**Cryptographic hash functions (such as SHA-256) are crucial for wallet security because they:

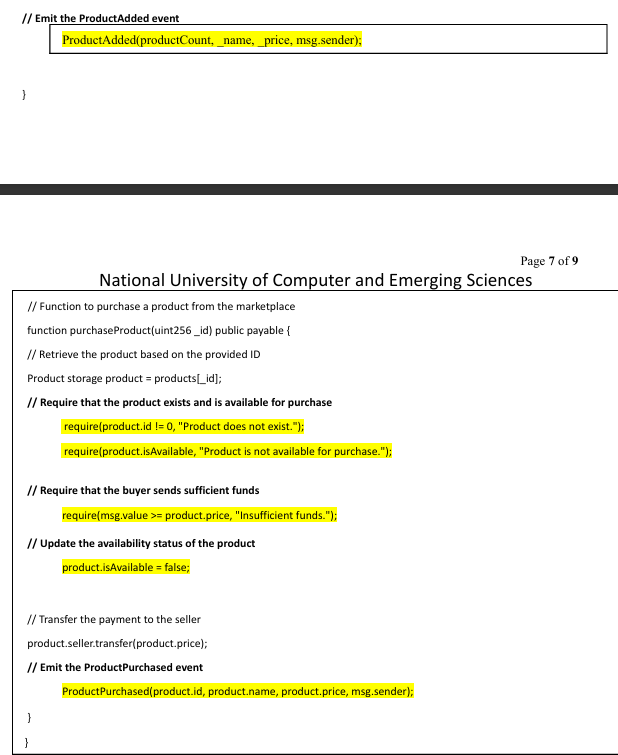
**Generate Fixed-Length, Unique Outputs:** Ensure that every public key, when hashed, produces a unique and fixed-length Bitcoin address.  
**Provide Pre-image Resistance:** It is computationally infeasible to derive the original public key from the Bitcoin address, protecting the actual key from being exposed.  
**Enable Data Integrity Verification:** Even a small change in the input results in a drastically different hash, allowing nodes to verify that transaction data hasn’t been altered. These properties together help maintain a high level of security for digital wallets, ensuring that even if an address is public, the underlying keys remain protected.

Past-Paper



Part 2:





23-26

Lecture 23

**HD Wallet:**

A **Hierarchically Deterministic (HD) Wallet** is a type of cryptocurrency wallet that uses a **single seed** to generate a **tree of key pairs** (public and private keys) using **standardized algorithms**.

Core idea is that we have one seed / parent key from which we can generate infinite public and priv keys.

-> A set of 12 or 24 words (mnemonic phrase) are generated

-> From the seed phrase the wallet creates a master priv key and a chain code.

🔒 **Benefits of HD Wallets:**

| **Feature** | **Description** |
| --- | --- |
| **Backup Simplicity** | One seed phrase backs up the whole wallet. |
| **Deterministic** | Same seed = same keys = same addresses. |
| **Security** | Can generate keys offline. |
| **Privacy** | Generates a new address for each transaction. |
| **Organization** | Allows multiple accounts or use-cases via branches. |

### **🔑 Key Features**

1. **Single Seed Backup**: All keys derive from one seed, simplifying backups.
2. **Master Public Key (MPK)**: Enables viewing of all wallet addresses and balances without access to private keys.
3. **Hierarchical Structure**: Supports generation of child keys from parent keys, allowing for organized key management.

## **✅ Advantages of Deterministic Wallets**

1. **Simplified Backup and Recovery**
   * Only the seed needs to be backed up.
   * Loss of the wallet file isn't catastrophic if the seed is preserved.
2. **Enhanced Privacy**
   * Ability to generate new addresses for each transaction, reducing address reuse.
3. **Organizational Flexibility**
   * Hierarchical wallets (like BIP32) allow for structured key derivation, useful for businesses managing multiple accounts or departments.
4. **Auditability**
   * MPK can be shared with auditors to verify transactions without exposing private keys.

## **⚠️ Understated Flaws and Vulnerabilities**

### **🔓 Key Derivation Vulnerability**

* Combining a **master public key (MPK)** with a **child private key** can compromise the **master private key**.
* This breaks the assumption that sharing MPK and individual private keys is safe.

### **🛠️ Exploit Demonstration**

* Using tools like *pybitcointools*, one can derive the master private key from an MPK and a corresponding child private key.
* This exploit undermines the security model of deterministic wallets when both MPK and child private keys are exposed

### **🧩 Implications**

* The practice of distributing child private keys to departments and MPKs to auditors is flawed.
* Such distribution can inadvertently expose the entire wallet if both elements are compromised.

**Crypto Wallet** :

A **crypto wallet** is a **digital tool** that lets you **store**, **send**, and **receive** cryptocurrencies like Bitcoin, Ethereum, etc. But it doesn't actually store coins - it stores your private keys , which give you access to your digital assets on the blockchain.

**Types of Crypto Wallet :**

1. Software Wallet
2. Hardware Wallet
3. Paper Wallet

**Software Wallet :**

**They are also known as Hot wallets**

It is further divided into 3 types

1. Desktop Wallet
2. Mobile Wallet
3. Online Wallet

**Desktop Wallet:**

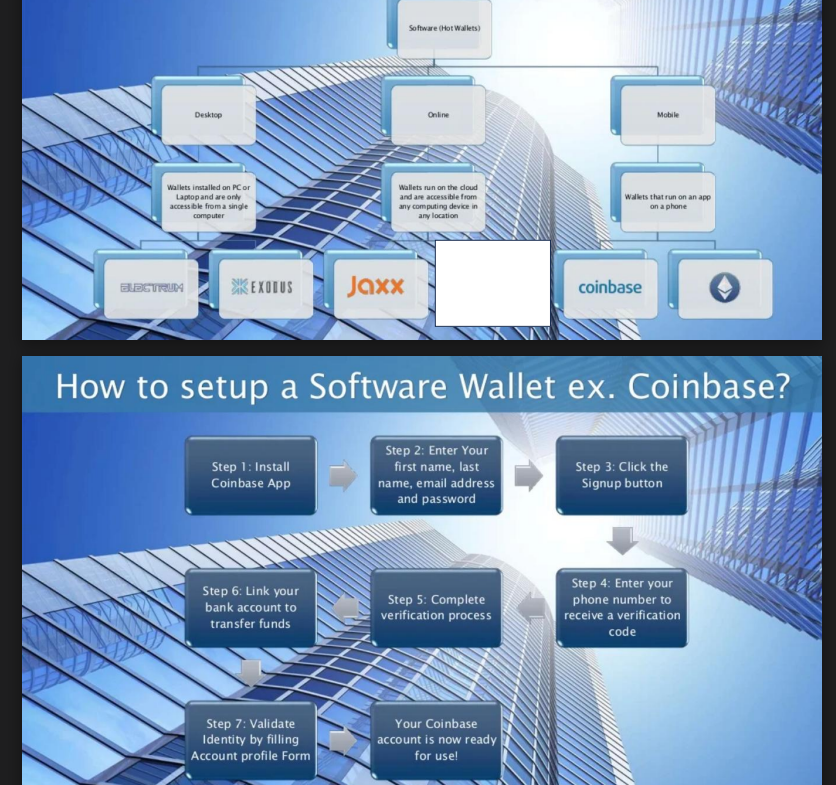
Desktop wallet is one which is downloaded and installed on a desktop or laptop. It can be only accessed in the system in which it is downloaded. Desktop Wallet also offers high end security until the system is hacked .When the system gets a virus there is the possibility that you may lose your crypto assets.

**Online Wallet :**

Online Wallet runs on cloud storage and is accessible from any device conveniently. Online Wallet stores private and public keys online which are controlled by third parties. This makes it more harmful and easily available for hackings and theft.

**Mobile Wallet :**

Mobile wallet is an app that runs on a mobile phone , which is more useful that they can be used anytime, anywhere. Mobile wallets are usually much smaller and simple as only limited space is available on the mobile.



Hot wallets are connected to internet , they are highly accessible , easier to use but they are less secure

**Hardware Wallet :**

A **hardware wallet** is a **physical device** (like a USB stick) used to **securely store your cryptocurrency private keys offline**. It is one of the **safest ways** to protect your crypto from online threats such as hacking, malware, or phishing attacks.

## **🧑‍💻 Technical Explanation**

* It **generates and stores your private key** offline.
* When you want to send crypto, the **transaction is signed inside the device**, then broadcasted to the blockchain from your computer or phone.
* Your **private key never touches the internet**, reducing the chance of theft.

🔒 **Key Features**

| **Feature** | **Description** |
| --- | --- |
| **Offline Key Storage** | Keep your private keys completely offline. |
| **Tamper-Proof** | Designed with secure chips to resist physical and software attacks. |
| **PIN & Passphrase** | Additional layers of protection. |
| **Transaction Signing** | Signs transactions within the device itself — private key never exposed. |

### **⚠️ Disadvantages**

* Can be expensive (compared to software wallets).
* If lost and not backed up (with seed phrase), **crypto is gone forever**.
* Slightly **less convenient** than hot wallets for quick use.

**Paper Wallet :**

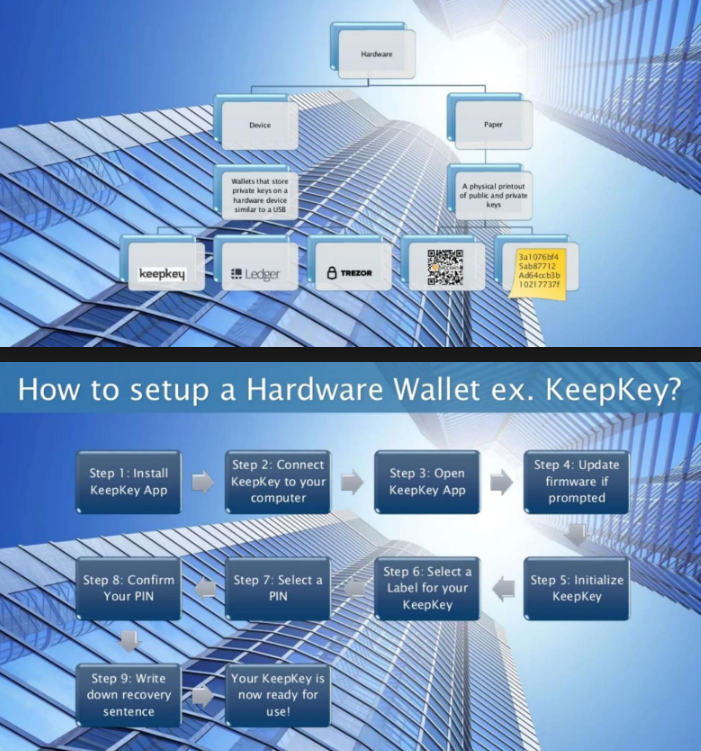
A **paper wallet** is a **physical piece of paper** that contains:

1. Your **private key**
2. Your **public address** (to receive crypto)  
    These are often printed as **QR codes** or long strings of characters.

It is an **offline cold storage** method used to keep your cryptocurrency safe from online hacks, viruses, and unauthorized access.

Pros and Cons :

They are extremely secure and does not involve internet but they are not quickly accessible



Lec 25-26

**Segregated Witness :**

**Segregated Witness (SegWit)** is a major protocol upgrade to Bitcoin, introduced in 2017 via **Bitcoin Improvement Proposal (BIP) 141**. It was designed to solve multiple issues in the Bitcoin network, most notably **scalability** and the **transaction malleability bug**.

**With SegWit:**

* The **witness data (signatures)** is separated from the main transaction data.
* This "segregated" data is stored in a new part of the block, allowing more transactions to fit.
* Block size **remains 1MB**, but the **effective block weight increases to 4MB**, boosting transaction throughput.

### **🚨 What Problem Did It Solve?**

1. **Transaction Malleability Fix:**
   * Previously, signatures could be slightly altered after a transaction was broadcast, changing its transaction ID.
   * SegWit removes the signature from the part of the transaction used to calculate the TXID, fixing this bug.
2. **Scalability:**
   * By removing bulky witness data, more transactions fit in a block.
   * This indirectly increases **transactions per second** (TPS).
3. **Foundation for Layer 2 (like Lightning Network):**
   * Fixing malleability was crucial for enabling smart contract-style systems on Bitcoin.

**DAOs :**

****

### 🧠 What Was The DAO?

* DAO = Decentralized Autonomous Organization.
* It was a kind of crowdfunded venture capital fund built on Ethereum.
* People sent ETH to the DAO smart contract, and in return, they got DAO tokens.
* These tokens let them vote on which projects the DAO should invest in.
* The DAO raised ~12 million ETH — worth $150 million at the time.

### **🔓 Why the DAO Hack Happened — The Core Flaw**

The vulnerability was due to a **re-entrancy bug** in the DAO smart contract.The root cause was a **vulnerability in the DAO smart contract code**, not a bug in Ethereum itself.

### **🔧 Technical Explanation (Re-Entrancy Bug)**

In Solidity (Ethereum’s smart contract language), the DAO’s smart contract let users **withdraw funds** by:

1. Sending ETH to the user.
2. **Then** updating the internal balance to show it was paid.

This was a mistake.

The hacker **created a malicious smart contract** that:

* Called the DAO’s withdraw() function.
* Received ETH.
* Before the DAO could update the balance, the malicious contract **called withdraw() again** — recursively.
* This cycle continued until **millions of ETH were drained**.

So the problem was:

**Updating the balance *after* sending ETH, instead of *before***

The solution to DAO attack was Hard Fork

### **🪓 What is a Hard Fork in Blockchain?**

A **hard fork** is a **permanent and backward-incompatible** change to a blockchain's rules or protocol. When a hard fork happens, the blockchain **splits into two separate chains** — one that follows the old rules, and one that follows the new rules.

### **🧠 Technical Explanation**

* In a **hard fork**, new **consensus rules** are introduced that are **not compatible** with the previous software.
* **Old nodes** (running outdated versions) **cannot validate** blocks created under the new rules.
* The result is a **split in the blockchain** into:  
  + The **original chain** (following old rules).
  + The **new chain** (following upgraded rules).

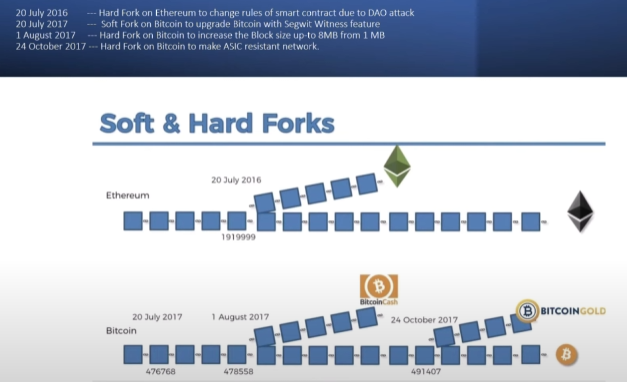
Both chains can continue independently, and **users may end up with coins on both chains** if the fork is contentious.

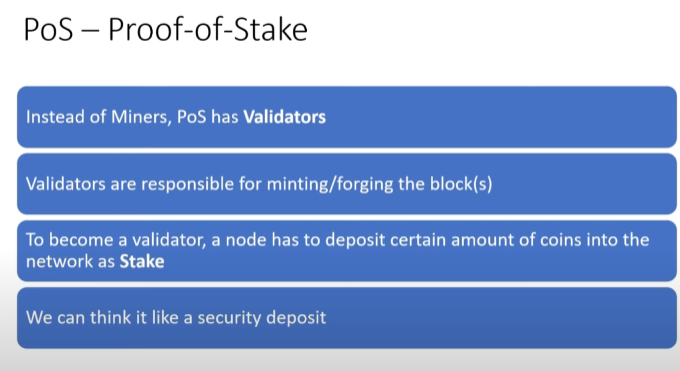
### **💥 Famous Examples**

| **Hard Fork** | **Result** |
| --- | --- |
| **Bitcoin Cash (2017)** | Split from Bitcoin to allow bigger block sizes (8MB vs 1MB) |
| **Ethereum Classic (2016)** | Split from Ethereum after the DAO hack. ETH reversed the hack, ETC didn't |

| **Feature** | **Hard Fork** | **Soft Fork** |
| --- | --- | --- |
| Compatibility | **Not backward compatible** | **Backward compatible** |
| Chain Split | Possible (if not all nodes upgrade) | No split (if most upgrade) |
| Rule Change | Enforces **new stricter or looser rules** | Enforces **stricter rules only** |
| Example | Bitcoin Cash, Ethereum Classic | SegWit (Bitcoin) |

SoftFork and HardFork on Bitcoin and Ethereum





## **1. 🧠 What is Proof of Stake?**

**Proof of Stake (PoS)** is a **consensus mechanism** used by blockchain networks to validate transactions and create new blocks — but instead of using computational power (as in Proof of Work), it uses **ownership (stake) of coins**.

In PoS, **the more coins you hold and lock up**, the higher your chance of being chosen to **validate the next block** and earn rewards.

## **3. 🎯 Why Do We Need It?**

Blockchain needs a way to:

* Agree on a **single version of the truth** (which block is next).
* Prevent **double spending**.
* Ensure that validators act honestly.

PoS offers:

* **Energy efficiency** (vs. mining in PoW).
* **Faster block confirmation**.
* Economic penalties for cheating.

## **4. ⚙️ How PoS Works (Step-by-Step)**

Let’s walk through the process:

### **Step 1: Stake**

* You **lock up your coins** in the network. This is called “staking.”
* The more you stake, the more skin you have in the game.

### **Step 2: Validator Selection**

* The network **randomly selects a validator**, often based on:  
  + Stake size
  + Coin age
  + Randomness

### **Step 3: Validate and Propose Block**

* The chosen validator:  
  + Collects pending transactions.
  + Verifies them.
  + Proposes a new block.

### **Step 4: Consensus and Reward**

* Other validators confirm the block is valid.
* The block is added to the chain.
* Validator gets a **reward** (new coins + transaction fees).

### **Step 5: Slashing (If Cheating)**

* If a validator tries to cheat or go offline:  
  + They can lose part or all of their stake.
  + This is called **slashing**.

## **5. 🧩 Key Components**

| **Term** | **Meaning** |
| --- | --- |
| **Stake** | Coins locked up as a deposit |
| **Validator** | User who proposes/validates blocks |
| **Delegator** | User who gives their coins to a validator (optional, like voting) |
| **Slashing** | Penalty for malicious behavior |
| **Reward** | Incentive for honest work |
| **Finality** | When a block can no longer be reversed |

## 

## 

## **6. ✅ Benefits of PoS**

* 💡 **Energy efficient** (no mining).
* ⚡ **Faster transactions**.
* 🏛️ **More decentralized** in some cases.
* 😇 **Less hardware requirement** — anyone with coins can participate.
* 🛡️ **Stronger economic penalties** for bad behavior.

## **7. ❌ Drawbacks / Criticisms**

* 🐋 **Wealth centralization**: The rich get richer.
* 🔒 **Nothing at Stake problem**: Validators could vote for multiple forks without cost (many PoS systems now solve this).
* 🎲 **Randomness bias**: Validator selection must be fair.
* 🧠 **Complexity**: More moving parts than PoW.

## **8. ⚖️ PoS vs PoW**

| **Feature** | **PoS (Proof of Stake)** | **PoW (Proof of Work)** |
| --- | --- | --- |
| Validation | By stake | By solving puzzles |
| Resource | Coins | Electricity + hardware |
| Energy use | Low | High |
| Hardware need | Low | High (ASICs, GPUs) |
| Attacker needs | 51% of stake | 51% of hash power |

## **9. 🔁 Variants of PoS**

| **Variant** | **Description** |
| --- | --- |
| **DPoS** (Delegated PoS) | Stakeholders vote for a few trusted validators (e.g., EOS, Tron) |
| **NPoS** (Nominated PoS) | Like DPoS but more decentralized (e.g., Polkadot) |
| **LPoS** (Leased PoS) | You lease your coins to validators (e.g., Waves) |
| **Hybrid PoW/PoS** | Combines both for security (e.g., Decred) |

## **10. 🔐 PoS Security**

### **Common Attacks:**

1. **Nothing at Stake Problem**:  
   * Solution: Slashing or finality rules.
2. **Long-Range Attacks**:  
   * Solution: Checkpoints.
3. **Bribery Attacks**:  
   * Solution: Randomized validator selection + slashing.

## **11. 🌐 Real-World PoS Blockchains**

| **Blockchain** | **Type** | **Year** | **Notes** |
| --- | --- | --- | --- |
| **Ethereum (ETH)** | PoS (since Sep 2022) | 2022 | Merged from PoW |
| **Cardano (ADA)** | PoS | 2017 | Ouroboros protocol |
| **Polkadot (DOT)** | NPoS | 2020 | Nominators + Validators |
| **Solana (SOL)** | PoS + Proof of History | 2020 | Super fast |
| **Tezos (XTZ)** | PoS | 2018 | Self-amending chain |

## **🧪 1. Casper – Ethereum’s PoS Protocol**

### **💡 What is Casper?**

**Casper** is the name for Ethereum's **Proof of Stake** consensus protocol family. The most well-known version is **Casper the Friendly Finality Gadget (FFG)**.

### **🛠️ How It Works**

* **Validators stake ETH** to get chosen to propose blocks.
* Casper focuses on **finality** — making blocks **irreversible** after enough validators agree.
* It combines:  
  + A **proposal mechanism** (beacon chain proposes blocks)
  + A **finality gadget** (validators “vote” on checkpoint blocks)

### **✅ Casper Strengths**

* Strong economic penalties for cheating
* Finality mechanism makes Ethereum more secure
* Scalable and energy-efficient

## **🌀 2. Ouroboros – Cardano’s PoS Protocol**

### **💡 What is Ouroboros?**

**Ouroboros** is Cardano’s scientifically peer-reviewed **Proof of Stake protocol**. It’s named after the mythical **snake eating its own tail**, symbolizing **self-sustaining cycles**.

### **🧠 How It Works**

1. **Time is divided into Epochs**, and each epoch is split into **slots**.
2. **Slot Leaders** are randomly chosen (based on stake) to create a block in that slot.
3. The protocol ensures randomness through a **verifiable random function (VRF)**.
4. Honest majority assumption: As long as the majority of stake is honest, the network is secure.

### **🔒 Security and Formal Proof**

* Ouroboros is the **first PoS protocol with mathematical security proofs**.
* It models attackers, network delay, and validator behavior.
* Variants:  
  + **Ouroboros Classic**
  + **Ouroboros Praos**
  + **Ouroboros Genesis** (handles dynamic validator sets)

### **✅ Ouroboros Strengths**

* Provable security like Bitcoin but using PoS
* Energy efficient
* Designed for long-term sustainability and upgradability

### **🔐 Proof of Authority (PoA)**

A **consensus mechanism** where a small number of **pre-approved, trusted validators** (usually known entities) are authorized to validate blocks and secure the network.

💡 *"Trust the identity, not the stake or computation."*

🧩 Used in: Private blockchains, Ethereum testnets (like Rinkeby)

### **🔥 Proof of Burn (PoB)**

A consensus mechanism where participants **“burn” (destroy) coins** by sending them to an unspendable address to earn the right to validate blocks or receive rewards.

💡 *"Prove your commitment by destroying your own coins."*

🧩 Used in: Slimcoin, Counterparty