Lecture Notes: Lesson 1

Blockchain Fundamentals

Cryptocurrency Course

Contents

1 Introduction to Blockchain Technology

1.1 Historical Context

The blockchain revolution began in 2008 with Satoshi Nakamoto's Bitcoin whitepaper titled *"Bitcoin: A Peer-to-Peer Electronic Cash System"*. This groundbreaking work solved the long-standing double-spending problem without requiring a trusted third party.

Key Innovation

Blockchain technology enables **trustless consensus** in a distributed network. Participants can agree on the state of a shared ledger without trusting any single authority or intermediary.

1.2 Core Characteristics

Blockchain systems exhibit several fundamental properties:

- **Decentralization**: No single point of control or failure
- **Immutability**: Historical records cannot be altered without detection
- **Transparency**: All transactions are visible to network participants
- **Security**: Cryptographic techniques protect data integrity
- **Consensus**: Agreement mechanisms ensure network-wide coordination

2 Block Structure and Components

2.1 Anatomy of a Block

Each block in a blockchain consists of two main parts: the **header** and the **body**.

2.1.1 Block Header

The header contains metadata that uniquely identifies and links the block:

1. **Block Hash**: A unique identifier computed from all header fields
2. **Previous Block Hash**: Links to the parent block, forming the chain
3. **Timestamp**: When the block was created (Unix epoch time)
4. **Nonce**: A variable used in the mining process (Proof-of-Work)
5. **Merkle Root**: A cryptographic digest of all transactions in the block

Block Hash Formula

The block hash is computed as:

$$\text{BlockHash} = \text{SHA256}(\text{SHA256}(\text{Header}))$$

This double-SHA256 provides additional security against potential collision attacks.

2.1.2 Block Body

The body contains the actual transaction data. In Bitcoin, a block can contain thousands of transactions. The Merkle root in the header provides a compact cryptographic summary of all these transactions.

2.2 Mathematical Representation

We can formally define a block B_i as:

$$B_i = (H_i, T_i)$$

where:

- H_i is the header containing: $\{h_{i-1}, t_i, n_i, m_i\}$
- h_{i-1} is the hash of the previous block
- t_i is the timestamp
- n_i is the nonce
- m_i is the Merkle root
- $T_i = \{tx_1, tx_2, \ldots, tx_k\}$ is the set of transactions

3 Cryptographic Hash Functions

3.1 Properties of Hash Functions

A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ must satisfy:

1. **Deterministic**: Same input always produces same output
2. **Fast Computation**: $H(m)$ can be computed quickly
3. **Pre-image Resistance**: Given h, it's infeasible to find m such that $H(m) = h$
4. **Second Pre-image Resistance**: Given m_1, it's infeasible to find $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$
5. **Collision Resistance**: It's infeasible to find any pair (m_1, m_2) such that $H(m_1) = H(m_2)$
6. **Avalanche Effect**: Small input changes produce dramatically different outputs

SHA-256 Example

Consider two similar inputs:

```
import hashlib

# Original input
input1 = "Hello World"
hash1 = hashlib.sha256(input1.encode()).hexdigest()
print(f"Hash 1: {hash1}")

# Slightly modified input
input2 = "Hello world"  # lowercase 'w'
hash2 = hashlib.sha256(input2.encode()).hexdigest()
print(f"Hash 2: {hash2}")
```

Output:

```
Hash 1: a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e
Hash 2: 64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c
```

Notice: A single character change produces a completely different hash!

3.2 SHA-256 Algorithm Overview

SHA-256 (Secure Hash Algorithm 256-bit) processes data in 512-bit blocks:

1. **Preprocessing**: Pad message to multiple of 512 bits
2. **Initialize Hash Values**: Eight 32-bit words (constants)
3. **Process Message Blocks**: 64 rounds of operations per block
4. **Output**: Final 256-bit hash value

The algorithm uses logical functions (AND, OR, XOR, NOT), bitwise rotation, and modular addition to achieve strong cryptographic properties.

4 Hash Chain and Blockchain Linkage

4.1 Chain Formation

The blockchain's immutability comes from its hash chain structure. Each block contains the hash of its predecessor, creating a cryptographic link:

$$B_0 \xrightarrow{h_0} B_1 \xrightarrow{h_1} B_2 \xrightarrow{h_2} B_3 \xrightarrow{h_3} \cdots$$

Immutability Proof

If an attacker modifies any transaction in block B_i:

1. The Merkle root m_i changes
2. The block hash h_i changes
3. Block B_{i+1} now has an invalid previous hash reference
4. All subsequent blocks must be recomputed
5. The attacker must redo all Proof-of-Work for blocks i through n
6. This becomes computationally infeasible for deeply buried blocks

4.2 Genesis Block

The first block in the chain (Block 0) has no predecessor. It's called the **genesis block** and is typically hardcoded into the blockchain software.

Bitcoin's genesis block (January 3, 2009) includes the famous message:

> *"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"*

5 Merkle Trees

5.1 Structure and Purpose

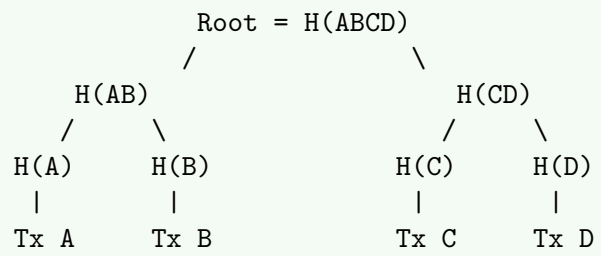
A Merkle tree is a binary tree of hashes used to efficiently summarize and verify large data sets.

5.1.1 Construction Algorithm

1. Hash each transaction: $H(tx_1), H(tx_2), \ldots, H(tx_n)$
2. Pair hashes and concatenate: $H(H(tx_1)||H(tx_2))$
3. Continue until single root hash remains
4. If odd number of leaves, duplicate the last hash

Merkle Tree Construction

For 4 transactions A, B, C, D:



Python implementation:

```python
import hashlib

def hash_pair(a, b):
    """Hash concatenation of two values"""
    return hashlib.sha256((a + b).encode()).hexdigest()

def build_merkle_root(transactions):
    """Build Merkle root from list of transactions"""
    if len(transactions) == 0:
        return None
    if len(transactions) == 1:
        return hashlib.sha256(transactions[0].encode()).hexdigest()

    # Hash all transactions
    level = [hashlib.sha256(tx.encode()).hexdigest()
             for tx in transactions]

    # Build tree bottom-up
    while len(level) > 1:
        next_level = []
        for i in range(0, len(level), 2):
            if i + 1 < len(level):
                next_level.append(hash_pair(level[i], level[i+1]))
            else:
                # Odd number: duplicate last hash
                next_level.append(hash_pair(level[i], level[i]))
        level = next_level

    return level[0]
```

5.2 Merkle Proofs

Merkle trees enable efficient proof of inclusion. To prove a transaction is in a block, you only need:

- The transaction itself
- $\log_2(n)$ sibling hashes (the Merkle path)
- The Merkle root from the block header

Complexity: For a block with 2048 transactions, you need only 11 hashes to prove inclusion (versus downloading all 2048 transactions).

6 Consensus Mechanisms

6.1 The Byzantine Generals Problem

Blockchain consensus solves a classic distributed systems problem: how can nodes agree on a single truth when some nodes may be malicious or faulty?

6.2 Proof-of-Work (PoW)

6.2.1 Mining Process

Miners compete to find a nonce such that:

$$H(\text{header}) < \text{target}$$

The target determines difficulty. Lower target = harder problem.

Difficulty Adjustment

Bitcoin adjusts difficulty every 2016 blocks to maintain approximately 10-minute block time:

$$\text{new\_difficulty} = \text{old\_difficulty} \times \frac{\text{expected\_time}}{\text{actual\_time}}$$

where expected time = 2016×10 minutes.

6.2.2 Security Analysis

An attacker needs $> 50\%$ of network hash rate to:

- Consistently outpace honest miners
- Rewrite transaction history
- Double-spend coins

Cost: Bitcoin's hash rate (as of 2024) exceeds 400 EH/s, making attacks economically infeasible.

6.3 Proof-of-Stake (PoS)

6.3.1 Core Concept

Validators are chosen to propose blocks based on their stake (coins locked as collateral).

Selection probability:

$$P(\text{validator } i) \propto \text{stake}_i$$

6.3.2 Advantages

- Energy efficient (no computational waste)
- Slashing: Malicious validators lose their stake
- Lower barrier to entry

6.3.3 Challenges

- **Nothing-at-Stake**: Validators can vote on multiple forks without cost
- **Long-Range Attacks**: Attackers with past stakes can rewrite history
- Solutions: Checkpointing, slashing conditions

6.4 Comparison Table

Feature	Proof-of-Work	Proof-of-Stake
Energy Use	Very High	Low
Security Basis	Computational power	Economic stake
Entry Barrier	Hardware cost	Token purchase
51% Attack Cost	Rent/buy 51% hash rate	Own 51% of supply
Finality	Probabilistic	Fast/deterministic
Examples	Bitcoin, Litecoin	Ethereum 2.0, Cardano

7 Decentralization Principles

7.1 Network Topology

Blockchain networks use peer-to-peer (P2P) architecture where:

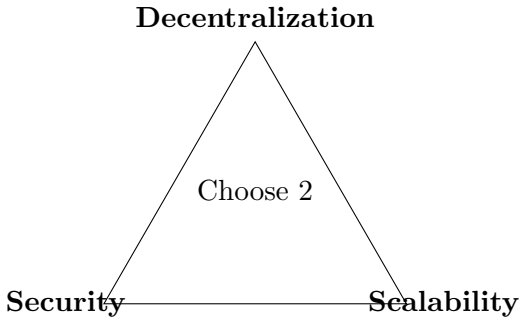
- Each node maintains a full or partial copy of the ledger
- Nodes communicate directly without central coordination
- Transaction propagation uses gossip protocol

7.2 Benefits of Decentralization

1. **Censorship Resistance**: No single entity can block transactions
2. **Fault Tolerance**: Network survives individual node failures
3. **Reduced Trust Requirements**: Cryptography replaces institutional trust
4. **Transparency**: All participants can audit the system

7.3 Decentralization Trilemma

Vitalik Buterin identified three desirable properties that are difficult to achieve simultaneously:



Trade-offs are inevitable; different blockchains optimize different corners of this triangle.

8 Study Questions

8.1 Conceptual Questions

1. Explain why changing a single transaction in block 100 would invalidate all subsequent blocks.
2. How does the Merkle root enable lightweight clients to verify transactions?
3. Compare the security models of Proof-of-Work versus Proof-of-Stake.
4. What prevents a miner from including invalid transactions in a block?
5. Describe the avalanche effect and its importance in cryptographic hashing.

8.2 Computational Problems

1. Given transactions [A, B, C, D, E, F, G, H], draw the complete Merkle tree and calculate the root hash (use simplified hash function for demonstration).
2. If Bitcoin's current difficulty requires finding a hash starting with 19 leading zeros, what is the approximate probability of success for a single hash attempt?
3. A blockchain has 100,000 blocks with 2000 transactions each. How many hashes are needed for a Merkle proof of inclusion?
4. Calculate the new difficulty if the last 2016 blocks took 18,000 minutes instead of the expected 20,160 minutes.

8.3 Implementation Exercises

1. Implement a simple blockchain with basic block structure (header + body).
2. Create a function to validate the hash chain integrity.
3. Build a Merkle tree implementation with proof generation and verification.
4. Simulate a basic Proof-of-Work mining process with adjustable difficulty.

9 Further Reading

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.*
- Antonopoulos, A. M. (2017). *Mastering Bitcoin: Programming the Open Blockchain.*
- Narayanan, A., et al. (2016). *Bitcoin and Cryptocurrency Technologies.*
- Ethereum Foundation. (2023). *Ethereum Whitepaper and Yellowpaper.*
- Buterin, V. (2017). *The Blockchain Scalability Trilemma.*