Fbc (4361603) - Problem Solver

Antigravity AI

June 4, 2026

Fbc

First Edition: 2026

Author: Antigravity AI
Website: milav.in
YouTube: @milav.dabgar

To my students,
who constantly inspire me to find new analogies,
break down complex theories, and make learning
an engineered science.

Preface

About this Book

Welcome to **Computer Networks**! This textbook is meticulously designed to serve as a comprehensive problem solver and study companion. It provides a robust and intuitive foundation in the core principles of this subject, bridging the gap between theoretical models and practical applications. A unique feature of this book is its focus on decoding complex concepts using clear, step-by-step methodologies and real-world analogies.

Target Audience

This book is intended for students and professionals looking to master the concepts of Computer Networks. Whether you are revising for exams, seeking to solidify your fundamental understanding, or looking for practical examples to clarify abstract concepts, this book has been structured to support your learning journey. It serves as an accessible guide designed to remove the fear of complex topics, replacing it with an appreciation for the underlying mechanics.

Scope and Coverage

The coverage is divided logically to follow standard academic curriculums. Each chapter focuses on key topics, breaking them down into digestible sections:

- **Theoretical Insights** – Core concepts explained intuitively.
- **Method Blocks** – Standardized, step-by-step procedures for solving specific types of problems.
- **Worked Examples** – Fully solved problems that apply the methods in practice.

We hope this resource proves invaluable in your studies and helps you achieve excellence in **Computer Networks**. Happy learning!

Acknowledgments

Writing a comprehensive book that connects the fundamental forces of Chemistry with practical engineering applications requires more than just academic knowledge—it requires a community of support. I would like to express my sincere gratitude to everyone who contributed to the realization of this project.

Specially, I extend my heartfelt gratitude to the **Department of Technical Education (DTE), Government of Gujarat**, and **Government Polytechnic, Palanpur**, for providing an environment that fosters innovation, academic rigor, and continuous professional development.

I am deeply thankful to my family for their unwavering patience and support during the countless hours spent researching, formatting, and refining these chapters.

Finally, my deepest appreciation goes to my students. Your curiosity, challenging questions, and continuous feedback are the true driving force behind the unique analogies and streamlined structure of this book. This textbook was engineered for you.

Antigravity AI

About the Author

Milav Jayeshkumar Dabgar is an engineering educator and R&D professional with over 9 years of experience in electronics hardware development, embedded systems, AI/ML, and full-stack software engineering. He currently serves as a **Lecturer (Class - II) & Innovation Leader** at Government Polytechnic, Palanpur, under the Education Department of the Government of Gujarat.

Milav holds a Master of Engineering (ME) in Communication Systems from L.D. College of Engineering and a Bachelor of Engineering (BE) in Electronics & Communication. He is also currently pursuing a **BS in Data Science and Applications from IIT Madras**, where he has completed both the **Diploma in Programming** and the **Diploma in Data Science** with distinction.

Most recently, he completed the **AICTE QIP PG Certificate Program in Deep Learning from SVNIT Surat** with a **perfect 10/10 CGPA**, topping his batch.

A dedicated mentor, Milav has guided numerous student teams in securing over **INR 45 lakhs** in innovation funding, including INR 25 lakhs from Shark Tank India and INR 20 lakhs through iHub Gujarat, resulting in two student patents. He is recognized as an **NPTEL Evangelist** and **NPTEL Discipline Star** in Computer Science, reflecting his commitment to continuous learning and academic excellence.

His technical expertise spans from embedded systems (8051, STM32, Arduino) to modern web technologies (Next.js, Python, PostgreSQL) and Data Science (TensorFlow, PyTorch). Through this textbook, he aims to simplify complex Engineering Chemistry concepts by integrating relatable, real-world analogies drawn from modern tech and engineering landscapes.

Milav is also an active content creator, running a popular **YouTube channel** (@milav.dabgar) where he shares technical tutorials and academic resources. He maintains a personal blog and resource hub at milav.in and can be reached for professional collaborations on LinkedIn.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction to Blockchain and Distributed Ledgers

1.1 Explain Blockchain and Distributed Ledger

A Blockchain is a decentralized and distributed ledger technology where data is stored in blocks that are cryptographically linked together. A Distributed Ledger is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, or institutions. From a computational perspective, the core of a blockchain relies on cryptographic hash functions and consensus algorithms to maintain the integrity of the ledger.

Methodology:

Cryptographic Hashing for Block Creation The fundamental computational operation in a blockchain is creating a cryptographic hash for a block. This process links blocks together securely.

1. **Gather Block Data**: Collect the data to be included in the block (D).
2. **Retrieve Previous Hash**: Obtain the hash of the preceding block (H_{prev}).
3. **Include a Nonce**: Select an arbitrary number called a nonce (N) used to vary the output hash.
4. **Concatenate Inputs**: Combine the previous hash, data, and nonce into a single string: $S = H_{prev} \parallel D \parallel N$.
5. **Apply Hash Function**: Pass the concatenated string through a cryptographic hash function (e.g. SHA-256) to produce the block hash: $H_{current} = \text{Hash}(S)$.

Worked Example:

Calculate a Simple Block Hash Suppose we have a simplified hash function $H(x) = (\sum \text{ASCII}(x)) \pmod{100}$. Given the previous block hash $H_{prev} =$ "12", block data $D =$ "TX1", and a nonce $N =$ "5". Calculate the new block hash.
Step 1: Concatenate the inputs Concatenate H_{prev}, D, and N: $S =$ "12TX15"
Step 2: Calculate the ASCII values of the characters in the string

- ASCII('1') = 49
- ASCII('2') = 50
- ASCII('T') = 84
- ASCII('X') = 88
- ASCII('1') = 49
- ASCII('5') = 53

Step 3: Sum the ASCII values Sum $= 49 + 50 + 84 + 88 + 49 + 53 = 373$
Step 4: Apply the modulo arithmetic $H_{current} = 373 \pmod{100} = 73$
The new block hash is "73".

Methodology:

Longest Chain Rule for Ledger Consensus In a distributed ledger, multiple nodes may mine a block simultaneously, leading to a fork. The Longest Chain Rule is a computational method to resolve such conflicts.

1. **Identify Branches**: Detect the divergent chains (branches) in the network.
2. **Calculate Chain Length**: Count the total number of blocks in each branch starting from the genesis block or the point of divergence.
3. **Select the Longest Chain**: The valid state of the ledger is the branch with the maximum length.
4. **Discard Orphan Blocks**: Blocks present in the shorter chains are discarded and their transactions are returned to the memory pool.

Worked Example:

Resolve a Ledger Fork A blockchain network experiences a fork at Block 50. Node A broadcasts a chain ending with Block 53A (Path: 50 → 51A → 52A → 53A). Node B broadcasts a chain ending with Block 52B (Path: 50 → 51B → 52B). Determine the valid chain according to the longest chain rule.
Step 1: Calculate the length of Node A's chain The length from the divergence point (inclusive of blocks added) is 3 blocks (51A, 52A, 53A). Total chain length $= 50 + 3 = 53$.
Step 2: Calculate the length of Node B's chain The length from the divergence point is 2 blocks (51B, 52B). Total chain length $= 50 + 2 = 52$.
Step 3: Apply the Longest Chain Rule Length of Chain A (53) > Length of Chain B (52).
Step 4: Conclusion The network will adopt Node A's chain as the canonical distributed ledger. Blocks 51B and 52B become orphan blocks.

1.2 Explain Asymmetric Key Encryption

Asymmetric key encryption, also known as public-key cryptography, is critical in blockchain for digitally signing transactions and ensuring data authenticity. It uses a pair of mathematically linked keys: a public key (shared openly) and a private key (kept secret). The RSA (Rivest-Shamir-Adleman) algorithm is a primary example of this computational approach.

Methodology:

RSA Algorithm for Key Generation and Encryption The RSA algorithm relies on the mathematical properties of large prime numbers.

1. **Choose Primes**: Select two distinct prime numbers p and q.
2. **Compute Modulus**: Calculate $n = p \times q$. The value n is used as the modulus for both public and private keys.
3. **Compute Totient**: Calculate Euler's totient function $\phi(n) = (p-1) \times (q-1)$.
4. **Select Public Exponent**: Choose an integer e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$. The public key is (e, n).
5. **Compute Private Exponent**: Calculate d as the modular multiplicative inverse of e modulo $\phi(n)$. This means $(d \times e) \pmod{\phi(n)} = 1$. The private key is (d, n).

6. **Encryption**: To encrypt a message M (where $M < n$), calculate the ciphertext $C \equiv M^e \pmod{n}$.
7. **Decryption**: To decrypt the ciphertext C, calculate the plaintext $M \equiv C^d \pmod{n}$.

Worked Example:

RSA Key Generation and Encryption Let prime numbers $p = 11$ and $q = 13$. Encrypt the message $M = 9$.
Step 1: Compute Modulus n $n = p \times q = 11 \times 13 = 143$
Step 2: Compute Totient $\phi(n)$ $\phi(n) = (11-1) \times (13-1) = 10 \times 12 = 120$
Step 3: Select Public Exponent e Choose $e = 7$ (since 7 is coprime to 120). Public Key is $(7, 143)$.
Step 4: Compute Private Exponent d Find d such that $(7 \times d) \pmod{120} = 1$. Using the Extended Euclidean Algorithm we find $d = 103$. (Verification: $7 \times 103 = 721$. $721 = 6 \times 120 + 1$, so $721 \pmod{120} = 1$). Private Key is $(103, 143)$.
Step 5: Encrypt the Message $M = 9$ $C \equiv M^e \pmod{n} = 9^7 \pmod{143}$. Calculating this using modular exponentiation: $9^2 = 81$ $9^4 = 81^2 = 6561 \equiv 126 \pmod{143}$ $9^6 \equiv 126 \times 81 = 10206 \equiv 53 \pmod{143}$ $9^7 \equiv 53 \times 9 = 477 \equiv 48 \pmod{143}$ The encrypted ciphertext is 48.

Methodology:

Elliptic Curve Cryptography Key Generation Elliptic Curve Cryptography (ECC) is extensively used in blockchains (like Bitcoin and Ethereum) because it offers equivalent security to RSA with significantly smaller key sizes. It is based on the algebraic structure of elliptic curves over finite fields.

1. **Define the Curve**: Select an elliptic curve defined by the equation $y^2 \equiv x^3 + ax + b \pmod{p}$, where p is a prime number.
2. **Select a Base Point**: Choose a base point $G = (x_G, y_G)$ on the curve.
3. **Choose Private Key**: Select a random integer k as the private key.
4. **Calculate Public Key**: Compute the public key P by multiplying the base point G by the private key k using elliptic curve point multiplication: $P = k \times G$.

Worked Example:

Calculate ECC Public Key Given an elliptic curve $y^2 \equiv x^3 + 2x + 2 \pmod{17}$, a base point $G = (5, 1)$, and a private key $k = 2$. Calculate the public key $P = 2G$.
Step 1: Point Doubling Formula To compute $2G$, we add the point G to itself $(G + G)$. First calculate the slope (s): $s \equiv \frac{3x_G^2 + a}{2y_G} \pmod{p}$
Step 2: Calculate the Slope $s \equiv \frac{3(5)^2+2}{2(1)} \pmod{17}$ $s \equiv \frac{75+2}{2} \pmod{17}$ $s \equiv \frac{77}{2} \pmod{17}$ Since $77 \equiv 9 \pmod{17}$, we need to find $9 \times 2^{-1} \pmod{17}$. The modular inverse of 2 modulo 17 is 9 (since $2 \times 9 = 18 \equiv 1 \pmod{17}$). $s \equiv 9 \times 9 \pmod{17} = 81 \pmod{17} = 13$.
Step 3: Calculate the New x-coordinate $x_P \equiv s^2 - 2x_G \pmod{p}$ $x_P \equiv 13^2 - 2(5) \pmod{17}$ $x_P \equiv 169 - 10 \pmod{17}$ $x_P \equiv 159 \pmod{17}$ $159 = 17 \times 9 + 6$, so $x_P = 6$.
Step 4: Calculate the New y-coordinate $y_P \equiv s(x_G - x_P) - y_G \pmod{p}$ $y_P \equiv 13(5-6) - 1 \pmod{17}$ $y_P \equiv 13(-1) - 1 \pmod{17}$ $y_P \equiv -14 \pmod{17}$ Since $-14 \equiv 3 \pmod{17}$, $y_P = 3$.
The public key point is $P = (6, 3)$.

1.3 Describe CAT Theorem in Blockchain

In the context of distributed systems and blockchain, the syllabus reference to the "CAT theorem" is widely recognized to refer to the **CAP theorem** (Consistency, Availability, Partition Tolerance). The CAP theorem is a fundamental concept for understanding the trade-offs involved in designing decentralized ledger networks.

The CAP theorem states that a distributed data store can simultaneously provide at most two out of the following three guarantees:

1. **Consistency (C)**: Every read receives the most recent write or an error.
2. **Availability (A)**: Every request receives a (non-error) response without the guarantee that it contains the most recent write.
3. **Partition Tolerance (P)**: The system continues to operate despite an arbitrary number of messages being dropped or delayed by the network between nodes.

Because blockchains operate over a network (which is inherently prone to partitions), Partition Tolerance (P) is mandatory. Therefore, blockchain architectures must computationally trade-off between Consistency and Availability.

Methodology:

Evaluating Blockchain using CAP Theorem To logically evaluate a blockchain network's position on the CAP theorem triangle:

1. **Verify Partition Tolerance**: Confirm if the system distributes data across multiple nodes that can experience communication failures. If yes, P is satisfied.
2. **Assess Node Responses**: Simulate a network partition. If a node cannot communicate with the rest of the network, determine how it responds to client read and write requests.
   - If it responds with potentially stale data, it favors **Availability (A)**. (System is AP).
   - If it refuses to respond or returns an error to prevent serving stale data, it favors **Consistency (C)**. (System is CP).
3. **Determine System Classification**: Based on the assessment, classify the distributed ledger as either CP or AP.

Worked Example:

Evaluating Bitcoin Network using CAP Evaluate the Bitcoin blockchain using the CAP evaluation method to determine its classification.

Step 1: Verify Partition Tolerance (P) Bitcoin nodes are spread globally over the internet. Network partitions (delays, dropped packets) occur frequently. The network continues to process blocks. Therefore, P is satisfied.

Step 2: Assess Node Responses during a Partition Suppose the network splits into two partitions N_1 and N_2. A client queries a node in N_1 for its balance. The node in N_1 will reply immediately using its local copy of the ledger even though N_2 might have processed newer transactions. The system provides a response rather than an error.

Step 3: Determine Classification Because the system responds with potentially stale data (favoring uptime over immediate universal agreement), it prioritizes Availability (A) over strong Consistency (C). Instead of immediate consistency, Bitcoin relies on *Eventual Consistency* (once the partition resolves, the longest chain rule will synchronize the nodes).

Conclusion Under the CAP theorem, the Bitcoin blockchain is classified as an **AP (Available and Partition Tolerant)** system.

CHAPTER 2

Structure of Blockchain

2.1 Examine the Type of Blockchain Architecture

Methodology:

Blockchain Architecture Selection Algorithm To determine the appropriate blockchain architecture (Public, Private or Consortium) for a given system, apply the following logical steps based on access control and participation requirements:

1. **Analyze Read Access**: Determine if the general public needs read access to the ledger or if it must be restricted.
2. **Analyze Write and Consensus Access**: Determine who is allowed to participate in the consensus process and validate transactions.
3. **Apply Selection Criteria**:
   - **Public Blockchain**: If read access is open to everyone AND consensus participation is permissionless (open to anyone).
   - **Private Blockchain**: If both read access and consensus participation are restricted to a single organization.
   - **Consortium Blockchain**: If consensus participation is restricted to a pre-selected set of nodes across multiple collaborating organizations.

Worked Example:

Architecture Selection for Supply Chain A global supply chain involves 5 distinct manufacturing companies. They need a shared ledger to track shipments. Only these 5 companies should be able to validate transactions, but the shipping records must be kept confidential from the general public. Determine the required blockchain architecture.

Solution:

1. **Step 1: Analyze Read Access**: The data must be kept confidential from the public. Therefore, read access is restricted.
2. **Step 2: Analyze Write and Consensus Access**: Exactly 5 distinct companies are allowed to validate transactions.
3. **Step 3: Apply Selection Criteria**: Since consensus is controlled by a pre-selected set of multiple organizations (not a single entity and not fully open), the appropriate architecture is a **Consortium Blockchain**.

2.2 Components of Blockchain

Methodology:

Merkle Root Calculation Method The Merkle Root is a fundamental component of a block header that mathematically summarizes all transactions in a block.

1. **List Transactions**: Let the transactions in the block be $T_1, T_2, \ldots, T_n$. If n is odd, duplicate the last transaction to make the count even.
2. **Hash Individual Transactions**: Calculate the hash of each transaction to form the leaf nodes: $H_i = \text{Hash}(T_i)$.
3. **Pair and Hash**: Concatenate adjacent pairs of hashes and hash the result to form the parent nodes: $H_{ij} = \text{Hash}(H_i + H_j)$.
4. **Repeat**: Continue pairing and hashing the nodes level by level until only one hash remains.
5. **Result**: The final single hash is the Merkle Root.

Worked Example:

Calculating a Merkle Root Given four transactions $T_1 =$ "A", $T_2 =$ "B", $T_3 =$ "C", $T_4 =$ "D". For simplicity in this manual calculation, assume our hash function $H(x)$ simply concatenates the inputs and wraps them in parentheses, e.g. $H(x) = (x)$ and $H(x + y) = (xy)$. Find the Merkle Root.
Solution:

1. **Step 1: List Transactions**: T_1, T_2, T_3, T_4 (count is 4, which is even).
2. **Step 2: Hash Individual Transactions (Leaf Nodes)**:
   - $H_1 = H(T_1) = (A)$
   - $H_2 = H(T_2) = (B)$
   - $H_3 = H(T_3) = (C)$
   - $H_4 = H(T_4) = (D)$
3. **Step 3: Pair and Hash (Level 1)**:
   - $H_{12} = H(H_1 + H_2) = ((A)(B))$
   - $H_{34} = H(H_3 + H_4) = ((C)(D))$
4. **Step 4: Pair and Hash (Level 2)**:
   - $H_{1234} = H(H_{12} + H_{34}) = (((A)(B))((C)(D)))$
5. **Step 5: Result**: The Merkle Root is $(((A)(B))((C)(D)))$.

2.3 Apply Concept of Asymmetric Key Encryption

Methodology:

RSA Asymmetric Key Generation and Encryption RSA is a standard asymmetric encryption algorithm used to generate key pairs for securing blockchain transactions.

1. **Key Generation**:
   (a) Choose two distinct prime numbers p and q.
   (b) Calculate the modulus $n = p \times q$.

(c) Calculate Euler's totient function $\phi(n) = (p-1) \times (q-1)$.

(d) Choose a public exponent e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The public key is (e, n).

(e) Calculate the private exponent d such that $(d \times e) \pmod{\phi(n)} = 1$. The private key is (d, n).

2. **Encryption**: Given a plaintext message M (where $M < n$), calculate the ciphertext C using the recipient's public key:
$$C = M^e \pmod{n}$$

3. **Decryption**: Given a ciphertext C, calculate the plaintext M using the recipient's private key:
$$M = C^d \pmod{n}$$

Worked Example:

RSA Encryption and Decryption Generate RSA keys using $p = 3$ and $q = 11$. Encrypt the message $M = 4$ and then decrypt the resulting ciphertext to verify the algorithm.
Solution:

1. **Key Generation**:
   - $p = 3$, $q = 11$.
   - $n = p \times q = 3 \times 11 = 33$.
   - $\phi(n) = (p-1) \times (q-1) = 2 \times 10 = 20$.
   - Choose e coprime to 20. Let $e = 3$. (Since $\gcd(3, 20) = 1$). Public Key $= (3, 33)$.
   - Find d such that $(d \times 3) \pmod{20} = 1$. Testing values: $d = 7$ gives $7 \times 3 = 21 \equiv 1 \pmod{20}$. Private Key $= (7, 33)$.

2. **Encryption**:
   - Message $M = 4$.
   - $C = M^e \pmod{n} = 4^3 \pmod{33}$.
   - $C = 64 \pmod{33} = 31$.
   - Ciphertext $C = 31$.

3. **Decryption**:
   - $M' = C^d \pmod{n} = 31^7 \pmod{33}$.
   - To simplify $31^7 \pmod{33}$, note that $31 \equiv -2 \pmod{33}$.
   - $M' = (-2)^7 \pmod{33} = -128 \pmod{33}$.
   - $-128 \pmod{33} = -128 + 4(33) = -128 + 132 = 4$.
   - Decrypted message $M' = 4$, which correctly matches the original message M.

2.4 Decentralized Network

Methodology:

Byzantine Fault Tolerance Threshold Calculation In a decentralized network, achieving consensus requires the system to be resilient against malicious (Byzantine) nodes. Practical Byzantine Fault Tolerance (PBFT) relies on specific node thresholds.

1. **Identify Malicious Nodes (f)**: Determine the maximum number of nodes that can act maliciously or fail.

2. **Calculate Minimum Total Nodes (N)**: The total number of nodes required in the network to achieve consensus must satisfy:
$$N \geq 3f + 1$$

3. **Calculate Required Quorum (Q)**: To confirm a transaction, the network needs a minimum number of valid responses:
$$Q = 2f + 1$$

Worked Example:

Determining Node Requirements for PBFT A decentralized banking network uses a PBFT consensus mechanism. The network administrators anticipate that up to 4 nodes might be compromised by hackers simultaneously. Calculate the minimum total number of nodes required in the network and the number of votes needed to reach a consensus.
Solution:

1. **Step 1: Identify malicious nodes**: The maximum number of compromised nodes is $f = 4$.

2. **Step 2: Calculate minimum total nodes (N)**:

   - $N \geq 3f + 1$
   - $N \geq 3(4) + 1$
   - $N \geq 12 + 1 = 13$

   The network must have at least 13 nodes.

3. **Step 3: Calculate required votes (Q)**:

   - $Q = 2f + 1$
   - $Q = 2(4) + 1$
   - $Q = 8 + 1 = 9$

   To safely confirm a block, 9 honest votes are required.

Methodology:

Gossip Protocol Message Propagation In decentralized blockchain networks, transactions propagate using the Gossip protocol. During each round, every informed node sends the message to k randomly chosen peers.

1. **Define Parameters**: Total nodes N, and fanout k (number of peers each node contacts per round).

2. **Identify Informed Nodes**: Let I_r be the number of informed nodes at round r. Initially, $I_0 = 1$.

3. **Apply Ideal Propagation Formula**: Assuming no overlapping targets in early rounds, the number of informed nodes grows exponentially:
$$I_r = I_{r-1} + (k \times I_{r-1}) = I_{r-1} \times (1 + k) = (1 + k)^r$$

4. **Calculate Minimum Rounds Required (R)**: To find the minimum rounds required to inform all N nodes (assuming perfect non-overlapping fanout):
$$R = \lceil \log_{1+k}(N) \rceil$$

Worked Example:

Gossip Protocol Propagation Rounds A decentralized network contains 1000 nodes. A new transaction is generated by one node and propagated using a gossip protocol with a fanout of $k = 4$. Assuming ideal non-overlapping message delivery, calculate the number of rounds required for all nodes to receive the transaction.

Solution:

1. **Step 1: Identify Parameters**: Total nodes $N = 1000$, fanout $k = 4$.
2. **Step 2: Apply the Rounds Formula**:
   - $R = \lceil \log_{1+k}(N) \rceil$
   - $R = \lceil \log_{1+4}(1000) \rceil = \lceil \log_5(1000) \rceil$
3. **Step 3: Evaluate Logarithm**:
   - $5^3 = 125$
   - $5^4 = 625$
   - $5^5 = 3125$

   Since 1000 falls between 5^4 and 5^5, the value of $\log_5(1000)$ is approximately 4.29.
4. **Step 4: Apply Ceiling Function**:
   - $R = \lceil 4.29 \rceil = 5$

   It will theoretically take 5 rounds for the transaction to propagate to all 1000 nodes.

CHAPTER 3

Essentials of Blockchain

3.1 Explain consensus mechanism in Blockchain

Consensus mechanisms are protocols that ensure all participating nodes in a distributed network agree on a single state of the ledger. In a decentralized environment without a central authority, computational methods are required to reach an agreement even if some nodes fail or act maliciously.

Methodology:

Byzantine Fault Tolerance Calculation Byzantine Fault Tolerance (BFT) determines the maximum number of malicious or failing nodes a network can mathematically handle while still successfully reaching consensus.

1. **Identify Total Nodes (N):** Count the total number of participating nodes in the consensus network.
2. **Apply the BFT Inequality:** The system requires a minimum number of total nodes defined by $N \geq 3f + 1$ to reach consensus, where f represents the number of faulty or malicious nodes.
3. **Calculate Maximum Faulty Nodes (f):** Rearrange the inequality to compute the maximum allowed Byzantine nodes:
$$f = \lfloor \frac{N-1}{3} \rfloor$$
4. **Calculate Required Quorum:** The minimum number of honest, responsive nodes required to reach a binding agreement is given by:
$$Quorum = 2f + 1$$

Worked Example:

Evaluating Network Consensus Viability A private blockchain network consists of 16 validator nodes running a Practical Byzantine Fault Tolerance (PBFT) consensus algorithm. Compute the maximum number of malicious nodes the network can tolerate, and find the minimum number of honest nodes required to reach consensus.

Solution:

1. **Identify Total Nodes:** $N = 16$.
2. **Calculate Maximum Faulty Nodes (f):** Using the BFT formula:
$$f = \lfloor \frac{N-1}{3} \rfloor$$
$$f = \lfloor \frac{16-1}{3} \rfloor = \lfloor \frac{15}{3} \rfloor = 5$$
The network can tolerate up to 5 malicious or offline nodes.
3. **Calculate Minimum Honest Nodes (Quorum):** The minimum number of nodes required to agree is:
$$Quorum = 2f + 1 = 2(5) + 1 = 11$$

Alternatively, this can be calculated as total nodes minus the maximum allowed faulty nodes: $16 - 5 = 11$.

3.2 Describe Proof of Work (PoW) and block reward

Proof of Work (PoW) is a consensus algorithm that requires miners to solve a computational puzzle. Miners must repeatedly generate hashes until they find an output value that is lower than a specific target. The first miner to find a valid hash is granted the right to add the block and receives a mathematically scheduled block reward.

Methodology:

Proof of Work Difficulty and Hash Target Calculation

1. **Determine the Target (T):** The target is a massive number (usually 256-bit). A valid block's hash must be strictly less than or equal to this target.
2. **Relate Difficulty to Target:** The current Difficulty (D) represents how much harder it is to find a hash compared to the easiest possible difficulty. It is the ratio of the maximum possible target (T_{max}) to the current target (T):
$$D = \frac{T_{max}}{T} \implies T = \frac{T_{max}}{D}$$
3. **Calculate Probability of a Valid Hash (p):** For an n-bit hash function, the probability of a single hash being valid is:
$$p = \frac{T}{2^n}$$
4. **Calculate Expected Hashes (H_{exp}):** The expected number of hashes required to find a valid block is the reciprocal of the probability:
$$H_{exp} = \frac{1}{p} = \frac{2^n}{T}$$

Worked Example:

Calculating Required Hashes for PoW Assume a simplified blockchain system where the hash output length is 32 bits. The maximum possible target T_{max} is set to 2^{30}. If the network automatically adjusts the current network difficulty D to 4, compute the current target and the expected number of hashes a miner needs to calculate to find a valid block.

Solution:

1. **Calculate the Current Target (T):**
$$T = \frac{T_{max}}{D} = \frac{2^{30}}{4} = \frac{2^{30}}{2^2} = 2^{28}$$
2. **Calculate the Probability of Success (p):** The total number of possible hash values for a 32-bit system is 2^{32}.
$$p = \frac{T}{2^{32}} = \frac{2^{28}}{2^{32}} = 2^{-4} = \frac{1}{16} = 0.0625$$
3. **Calculate Expected Hashes (H_{exp}):**
$$H_{exp} = \frac{1}{p} = \frac{1}{1/16} = 16 \text{ hashes}$$

Methodology:

Block Reward Halving Mechanism Block rewards are given to successful miners to incentivize network participation. Many PoW blockchains systematically reduce this reward over time to control token inflation.

1. **Identify Initial Parameters:** Find the initial block reward given at the Genesis block (R_0) and the halving interval (I) expressed as a number of blocks.
2. **Determine the Halving Epoch (n):** For any given block height (h), the number of times the reward has been halved is calculated using the floor function:

$$n = \lfloor \frac{h}{I} \rfloor$$

3. **Calculate the Current Reward (R_h):** The block reward at block height h is:

$$R_h = \frac{R_0}{2^n}$$

Worked Example:

Determining Block Reward at Specific Height A blockchain starts with an initial block reward of 50 coins. The protocol is designed to halve the reward every 210,000 blocks. Calculate the block reward for a miner who successfully mines block number 650,000.

Solution:

1. **Identify Parameters:** Initial Reward $R_0 = 50$, Halving Interval $I = 210,000$, Block Height $h = 650,000$.
2. **Determine the Halving Epoch (n):**

$$n = \lfloor \frac{650,000}{210,000} \rfloor = \lfloor 3.095 \rfloor = 3$$

   The reward has halved exactly 3 times.

3. **Calculate Current Reward (R_h):**

$$R_{650000} = \frac{50}{2^3} = \frac{50}{8} = 6.25 \text{ coins}$$

To visualize the halving schedule, we can construct the following table mapping the epoch to the reward:

Epoch (n)	Block Height Range	Block Reward (R_h)
0	0 to 209,999	50.00
1	210,000 to 419,999	25.00
2	420,000 to 629,999	12.50
3	630,000 to 839,999	6.25

Since block 650,000 falls cleanly into Epoch 3, the reward computation of 6.25 coins is verified.

3.3 Examine type of attacks in Blockchain

Blockchains are susceptible to several attacks. The most notable computational attack is the 51% attack, where an entity gains majority control of the network's hash rate to reorganize the ledger, reverse transactions, and double-spend coins.

Methodology:

Attacker Success Probability Calculation This method calculates the probability that an attacker with a minority of the total hash rate can mathematically catch up to the honest network and replace the main chain.

1. **Identify Hash Rate Proportions:** Let p be the proportion of the network's hash rate controlled by honest nodes, and q be the proportion controlled by the attacker. Note that $p + q = 1$. Assume the attacker is a minority ($q < p$).
2. **Define the Block Deficit (z):** This is the number of blocks the attacker's secret chain is currently behind the main honest chain (often representing the number of confirmations a merchant waits for).
3. **Calculate Catch-up Probability (P_z):** The probability that the attacker will eventually generate enough blocks to catch up from z blocks behind and overtake the honest chain is given by:

$$P_z = \left(\frac{q}{p}\right)^z$$

Worked Example:

Probability of Attacker Catching Up An attacker controls 30% of a blockchain's total computational power, while honest miners control the remaining 70%. A merchant waits for 3 block confirmations before releasing digital goods. What is the probability that the attacker can successfully mine a longer secret chain to reorganize the blockchain and double-spend their coins?
Solution:

1. **Identify Proportions:** Attacker hash rate $q = 0.30$. Honest hash rate $p = 0.70$.
2. **Identify Deficit:** The merchant waits for 3 confirmations, meaning the honest chain is 3 blocks ahead. Therefore, $z = 3$.
3. **Calculate Probability:** Using the catch-up formula:

$$P_3 = \left(\frac{0.30}{0.70}\right)^3 = \left(\frac{3}{7}\right)^3$$

$$P_3 = \frac{27}{343} \approx 0.0787$$

The probability of the minority attacker successfully altering the transaction history is approximately 7.87%.

Methodology:

Analyzing 51 Percent Attack Cost To execute a guaranteed 51% attack, an adversary must rent or purchase enough hardware to strictly exceed the honest network's hash rate.

1. **Identify Honest Network Hash Rate (H_{net}):** Determine the current total hash rate of the honest network.
2. **Calculate Required Attacker Hash Rate (H_{att}):** To have a strict majority (over 50% of the newly combined network power), the attacker must deploy hash power slightly greater than H_{net}.

$$H_{att} > H_{net}$$

3. **Determine Renting Cost (C_{hourly}):** If the cost to rent hash power is C dollars per unit of hash rate per hour, the hourly cost to sustain the attack is:

$$C_{hourly} = H_{att} \times C$$

Worked Example:

Calculating the Cost of a 51 Percent Attack A minor cryptocurrency network operates with a total honest hash rate of $40,000$ Terahashes per second (TH/s). An attacker intends to launch a 51% attack by renting cloud mining equipment. The market cost to rent 1 TH/s of mining power is $0.15 per hour. Calculate the absolute minimum financial cost required to sustain this attack for a 24-hour period.

Solution:

1. **Determine Required Hash Rate:** To control more than 50% of the combined total hash rate, the attacker needs to add computational power equal to or slightly greater than the current honest network.

$$H_{att} = 40,001 \text{ TH/s}$$

2. **Calculate Hourly Cost:** Multiply the required hash rate by the cost per TH/s.

$$C_{hourly} = 40,001 \times 0.15 = \$6,000.15 \text{ per hour}$$

3. **Calculate Total Attack Cost for 24 Hours:**

$$C_{total} = C_{hourly} \times 24$$

$$C_{total} = 6,000.15 \times 24 = \$144,003.60$$

The attacker would require a minimum budget of $\$144,003.60$ to sustain the 51% attack for a full 24-hour day.

CHAPTER 4

Conceptualization of Blockchain as Cryptocurrency

4.1 Describe Bitcoin

The Bitcoin network operates as a decentralized peer-to-peer cryptocurrency system. The primary computational aspects involve managing Unspent Transaction Outputs (UTXO) for balances and solving the Proof of Work puzzle to mine new blocks.

Methodology:

Bitcoin UTXO Calculation The UTXO (Unspent Transaction Output) model determines user balances and transaction validity. The algorithm is as follows:

1. **Identify Inputs:** Gather existing UTXOs belonging to the sender that sum up to at least the required payment plus the transaction fee.
2. **Calculate Total Input:** Total Input $= \sum \text{UTXO}_i$.
3. **Define Outputs:** Specify the payment amount to the receiver.
4. **Calculate Change:** Change = Total Input − Payment Amount − Transaction Fee.
5. **Create New UTXOs:** The transaction creates new UTXOs for the receiver (Payment Amount) and the sender (Change).

Worked Example:

UTXO and Change Calculation Alice needs to send 4.5 BTC to Bob. Alice holds the following UTXOs in her wallet: UTXO1 = 2.0 BTC, UTXO2 = 1.5 BTC, UTXO3 = 3.0 BTC, and UTXO4 = 0.5 BTC. The network requires a transaction fee of 0.1 BTC. Determine the inputs used, the change returned to Alice, and the new UTXOs generated.

Step 1: Determine required total.

$$\text{Required} = \text{Payment} + \text{Fee} = 4.5 + 0.1 = 4.6 \text{ BTC}$$

Step 2: Select inputs. Alice's wallet selects UTXOs to meet or exceed 4.6 BTC. Selected: UTXO1 (2.0) + UTXO3 (3.0) = 5.0 BTC. (Other combinations work, but this is optimal).

$$\text{Total Input} = 5.0 \text{ BTC}$$

Step 3: Calculate the change.

$$\text{Change} = \text{Total Input} - \text{Payment} - \text{Fee}$$

$$\text{Change} = 5.0 - 4.5 - 0.1 = 0.4 \text{ BTC}$$

Step 4: Identify new UTXOs. The transaction consumes UTXO1 and UTXO3. It generates two new UTXOs:

- 4.5 BTC UTXO locked to Bob's public key.
- 0.4 BTC UTXO locked to Alice's public key (Change).

Methodology:

Proof of Work Target Verification Bitcoin miners compete to find a nonce such that the hash of the block header is less than a specific target.

1. **Construct Block Header:** Combine Previous Block Hash, Merkle Root, Timestamp, Target Difficulty, and Nonce.
2. **Apply Hash Function:** Compute Hash = SHA256(SHA256(Block Header)).
3. **Verify Target:** The block is valid if and only if Hash $\leq$ Target.
4. **Difficulty Adjustment:** The target adjusts every 2016 blocks. New Target = Old Target $\times$ $\frac{\text{Actual Time of Last 2016 Blocks}}{2016 \times 10 \text{ minutes}}$.

Worked Example:

Evaluating Proof of Work Validity A miner proposes a new block. The network target is currently set to '0x0000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'. The miner hashes the block header with Nonce A and Nonce B:

- Hash A: '0x00010A2B...'
- Hash B: '0x0000A1B2...'

Determine which nonce provides a valid Proof of Work.
Step 1: Analyze the Target. The target requires the hash to start with at least four leading zero hex digits, and the fifth digit must be less than or equal to 'F'. Mathematically, the value must be strictly less than or equal to '0x0000FFFF...'.
Step 2: Compare Hash A to Target. Hash A starts with '0x0001...'. '0x00010A2B...' > '0x0000FFFF...' Hash A is greater than the target. Nonce A is invalid.
Step 3: Compare Hash B to Target. Hash B starts with '0x0000A...'. '0x0000A1B2...' < '0x0000FFFF...' Hash B is less than the target. Nonce B is valid and the block is accepted.

4.2 Describe Consistency and Fault Tolerance

Blockchain networks must maintain a consistent state across all nodes, even when some nodes fail or act maliciously.

Methodology:

Byzantine Fault Tolerance Calculation Practical Byzantine Fault Tolerance (pBFT) algorithms determine the maximum number of malicious or failing nodes a network can tolerate while still reaching consensus.

1. **Identify Total Nodes:** Let N be the total number of replica nodes in the network.
2. **Calculate Fault Tolerance:** The maximum number of faulty nodes f that can be tolerated is calculated as:
$$f = \lfloor \frac{N-1}{3} \rfloor$$
3. **Calculate Quorum:** The minimum number of nodes required to agree (consensus threshold) is:
$$Q = 2f + 1$$

Worked Example:

Determining BFT Thresholds A private consortium blockchain operates using a pBFT consensus mechanism with 16 validating nodes. Calculate the maximum number of Byzantine faults the system can tolerate and the required number of honest nodes needed to validate a block.

Step 1: Identify Total Nodes (N).

$$N = 16$$

Step 2: Calculate maximum faulty nodes (f).

$$f = \lfloor \frac{16-1}{3} \rfloor = \lfloor \frac{15}{3} \rfloor = 5$$

The network can tolerate up to 5 malicious or offline nodes.

Step 3: Calculate the required quorum (Q).

$$Q = 2f + 1 = 2(5) + 1 = 11$$

Therefore, at least 11 nodes must agree to successfully achieve consensus and append a new block.

Methodology:

Longest Chain Rule for Consistency In Proof of Work networks, temporary forks occur when two miners find a block simultaneously. The network resolves this to maintain consistency using the Longest Chain Rule.

1. **Track Branches:** Nodes keep track of multiple valid block branches.
2. **Calculate Chain Work:** Compute the total cumulative difficulty (often simplified as block height) for each branch.
3. **Select Main Chain:** The branch with the greatest cumulative computational work is designated as the main chain.
4. **Orphan Blocks:** Blocks in the shorter branch become "orphan" or "stale" blocks, and their transactions are returned to the mempool if not present in the main chain.

Worked Example:

Resolving a Blockchain Fork A node observes a blockchain fork at Block Height 100. Branch X mines Block 101_X and 102_X. Branch Y mines Block 101_Y, 102_Y, and 103_Y. Both branches have the same difficulty target per block. Determine the active chain and the fate of Block 102_X.

Step 1: Calculate the length of Branch X. Branch X ends at height 102. Length from fork = 2 blocks.

Step 2: Calculate the length of Branch Y. Branch Y ends at height 103. Length from fork = 3 blocks.

Step 3: Apply Longest Chain Rule. Branch Y (length 3) > Branch X (length 2). The node adopts Branch Y as the main consistent chain.

Step 4: Handle Orphaned Blocks. Blocks 101_X and 102_X are orphaned. The node discards them. Transactions uniquely inside 101_X and 102_X that are not in Branch Y are sent back to the mempool to be mined in future blocks.

4.3 Apply Concept of Cryptography

Cryptographic functions secure data integrity and authenticate users in a blockchain.

Methodology:

RSA Asymmetric Cryptography RSA is a foundational asymmetric algorithm for generating key pairs.

1. **Choose Primes:** Select two distinct prime numbers, p and q.

2. **Compute Modulus (n):** $n = p \times q$.
3. **Compute Totient ($\phi(n)$):** $\phi(n) = (p-1) \times (q-1)$.
4. **Select Public Exponent (e):** Choose e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. Public Key is (e, n).
5. **Compute Private Exponent (d):** Compute d such that $(d \times e) \pmod{\phi(n)} = 1$. Private Key is (d, n).
6. **Encrypt/Decrypt:** $C = M^e \pmod{n}$ and $M = C^d \pmod{n}$.

Worked Example:

RSA Key Generation and Encryption Given prime numbers $p = 5$ and $q = 11$, and a public exponent $e = 3$. Calculate the private key d, and encrypt the message $M = 9$.

Step 1: Compute n and $\phi(n)$.

$$n = p \times q = 5 \times 11 = 55$$

$$\phi(n) = (p-1) \times (q-1) = 4 \times 10 = 40$$

Step 2: Calculate private exponent d. We need d such that $(d \times 3) \pmod{40} = 1$. Testing values of d: If $d = 27$, then $27 \times 3 = 81$. $81 \pmod{40} = 1$. Thus, $d = 27$. The private key is $(27, 55)$.

Step 3: Encrypt the message $M = 9$.

$$C = M^e \pmod{n}$$

$$C = 9^3 \pmod{55}$$

$$9^3 = 729$$

$$729 \div 55 = 13 \text{ with a remainder of } 14$$

$$C = 14$$

The encrypted ciphertext is 14.

Methodology:

Digital Signature Generation and Verification Digital signatures prove message authenticity and non-repudiation.

1. **Hash Message:** Sender computes $H = \text{Hash}(M)$.
2. **Sign:** Sender encrypts H with their Private Key (d) to create Signature $S = H^d \pmod{n}$.
3. **Transmit:** Sender sends (M, S) to Receiver.
4. **Verify Step 1:** Receiver decrypts S using Sender's Public Key (e): $H' = S^e \pmod{n}$.
5. **Verify Step 2:** Receiver computes hash of received message: $H'' = \text{Hash}(M)$.
6. **Authentication:** If $H' = H''$, the signature is valid.

Worked Example:

Verifying a Digital Signature Alice sends a transaction M with hash $H = 4$ to Bob. Alice's RSA keys are $(e = 7, n = 33)$ and $(d = 3, n = 33)$. Alice sends the signature S. Show how Alice creates S and how Bob verifies it.

Step 1: Alice creates the signature. Alice signs the hash $H = 4$ using her private key $d = 3$.

$$S = H^d \pmod{n} = 4^3 \pmod{33} = 64 \pmod{33}$$

$$64 = 33 \times 1 + 31$$

$$S = 31$$

Alice sends message M and signature $S = 31$.
Step 2: Bob verifies the signature. Bob receives $S = 31$ and uses Alice's public key $e = 7, n = 33$ to find H'.

$$H' = S^e \pmod{n} = 31^7 \pmod{33}$$

Notice that $31 \equiv -2 \pmod{33}$.

$$(-2)^7 = -128$$

$$-128 \pmod{33} \rightarrow -128 + 4(33) = -128 + 132 = 4$$

$$H' = 4$$

Step 3: Bob compares hashes. Bob independently hashes M to get $H'' = 4$. Since $H' = H''$ $(4 = 4)$, the signature is valid.

4.4 Describe Implementations of Blockchain in Cryptocurrency

Real-world cryptocurrency implementations rely on data structures like Merkle Trees to store transactions efficiently and mechanisms like Gas to allocate computational resources.

Methodology:

Merkle Tree Root Calculation A Merkle Tree is a binary tree of hashes used to efficiently summarize and verify the integrity of large sets of transactions in a block.

1. **Hash Transactions:** Compute the hash of each individual transaction to form the leaf nodes (H_A, H_B, H_C, H_D).
2. **Duplicate Odd Node:** If there is an odd number of nodes at any level, duplicate the last node.
3. **Pair and Hash:** Concatenate adjacent hash pairs and compute a new hash for the parent node: $H_{AB} = \text{Hash}(H_A + H_B)$.
4. **Repeat to Root:** Continue pairing and hashing up the tree levels until only a single hash remains. This is the Merkle Root.

Worked Example:

Constructing a Merkle Root A block contains 3 transactions: T_1, T_2, and T_3. Assume a simple hash function where $\text{Hash}(T_1) = A$, $\text{Hash}(T_2) = B$, and $\text{Hash}(T_3) = C$. The hash function simply concatenates inputs for this example: $\text{Hash}(X + Y) = XY$. Determine the Merkle Root.
Step 1: Hash the leaves. Leaf nodes are A, B, and C.
Step 2: Handle odd number of leaves. Since there are 3 leaves, duplicate the last leaf C. The working set is now: A, B, C, C.
Step 3: Pair and hash the first level. Pair 1: A and $B \rightarrow \text{Hash}(A + B) = AB$ Pair 2: C and $C \rightarrow \text{Hash}(C + C) = CC$ The next level contains nodes AB and CC.
Step 4: Pair and hash to find the root. Pair the remaining nodes: AB and $CC \rightarrow \text{Hash}(AB + CC) = ABCC$. The Merkle Root is $ABCC$.

Methodology:

Ethereum Gas Fee Calculation Ethereum utilizes a gas model to prevent infinite loops and allocate network resources. Transactions require a fee paid in Ether (ETH).

1. **Identify Gas Used:** Determine the total computational units consumed by the transaction (e.g., standard transfer $= 21,000$ gas).

2. **Identify Gas Price:** Determine the price the user is willing to pay per unit of gas, typically expressed in Gwei (1 Gwei $= 10^{-9}$ ETH).
3. **Calculate Total Fee:** Transaction Fee = Gas Used $\times$ Gas Price.
4. **Convert to ETH:** Multiply the Gwei result by 10^{-9} to express the fee in Ether.

Worked Example:

Calculating Smart Contract Execution Fees A user executes a smart contract function on the Ethereum network. The function consumes $45,000$ gas. The current network gas price is 30 Gwei. Calculate the total transaction fee in Gwei and in ETH.
Step 1: Identify variables. Gas Used $= 45,000$ Gas Price $= 30$ Gwei
Step 2: Calculate total fee in Gwei.

$$\text{Fee} = \text{Gas Used} \times \text{Gas Price}$$

$$\text{Fee} = 45,000 \times 30 = 1,350,000 \text{ Gwei}$$

Step 3: Convert the fee to ETH. Since 1 Gwei $= 10^{-9}$ ETH:

$$\text{Fee in ETH} = 1,350,000 \times 10^{-9}$$

$$\text{Fee in ETH} = 0.00135 \text{ ETH}$$

The user will pay a fee of 0.00135 ETH for executing the smart contract.

CHAPTER 5

Decentralization using Blockchain

Decentralization is the core paradigm shift introduced by blockchain technology, enabling peer-to-peer interactions without central intermediaries. This chapter focuses on the computational aspects of smart contracts and quantitative measures of decentralization in blockchain applications.

5.1 Describe Smart Contract

A smart contract is a self-executing program deployed on a blockchain network. It automatically executes predefined actions when specific computational conditions are met.

Methodology:

Smart Contract Execution Algorithm The execution of a smart contract follows a deterministic algorithmic process:

1. **Transaction Initiation**: A user broadcasts a transaction to the smart contract address containing input data and an execution fee (Gas).
2. **Validation**: Network nodes validate the transaction signature and ensure the sender has sufficient balance to cover the maximum gas limit.
3. **State Loading**: The virtual machine (e.g. EVM) loads the current state of the smart contract from the blockchain.
4. **Instruction Execution**: The virtual machine executes the compiled bytecode sequentially. Each opcode consumes a predefined amount of gas.
5. **State Transition**: If execution succeeds without running out of gas, the new state is saved, and remaining gas is refunded. If an error occurs or gas runs out, execution reverts, and all gas is consumed.

Worked Example:

Simple Escrow Contract Execution **Problem Statement:** An escrow smart contract is designed to release 500 Tokens to a `Seller` only if the `Buyer` confirms receipt of goods. If the `Buyer` does not confirm within 7 days, the contract automatically refunds the `Buyer`. Given the current blockchain timestamp is $T = 1670000000$ and the purchase was initiated at $T_{start} = 1669000000$ (where 7 days = 604800 seconds), determine the state transition if the `Buyer` invokes the `Refund()` function.
Solution:

1. **Identify Inputs:** Current Timestamp (T) = 1670000000
   Start Timestamp (T_{start}) = 1669000000
   Threshold for Refund = $T_{start} + 604800 = 1669604800$
2. **Evaluate Condition:** The `Refund()` function algorithm checks if $T >$ Threshold. Is $1670000000 > 1669604800$? Yes.
3. **Determine State Transition:** Since the time condition is met, the contract logic executes the

transfer from the contract balance to the `Buyer`'s address.

4. **Final Output:** The transaction is valid. 500 Tokens are transferred to the `Buyer`, and the contract state updates its balance to 0.

Methodology:

Gas Fee Calculation Formula To prevent infinite loops and allocate network resources accurately, smart contracts utilize a gas system. The computational cost is evaluated as follows:

1. **Gas Used (G_u):** The total sum of gas units consumed by the opcodes executed in the smart contract.
2. **Gas Price (P_g):** The price the user is willing to pay per unit of gas, usually measured in Gwei (1 Gwei $= 10^{-9}$ Ether).
3. **Total Transaction Fee (F):** Calculated using the formula:

$$F = G_u \times P_g$$

Worked Example:

Calculating Smart Contract Execution Cost **Problem Statement:** A decentralized voting application requires a user to interact with a smart contract to cast a vote. The transaction consumes 45000 units of gas. The current network gas price is 20 Gwei. Calculate the total transaction fee in Gwei and Ether.
Solution:

1. **Identify Variables:** Gas Used (G_u) $= 45000$
   Gas Price (P_g) $= 20$ Gwei
2. **Compute Total Fee in Gwei:**

$$F = G_u \times P_g = 45000 \times 20 = 900000 \text{ Gwei}$$

3. **Convert to Ether:** Since 1 Gwei $= 10^{-9}$ Ether:

$$F = 900000 \times 10^{-9} = 0.0009 \text{ Ether}$$

4. **Conclusion:** The execution of the voting smart contract will cost the user 0.0009 Ether.

5.2 Describe Decentralization in Blockchain Application

Decentralization in blockchain applications (DApps) ensures that control and decision-making are distributed among a network rather than concentrated in a single centralized entity. Computationally, we measure decentralization to determine network resilience against attacks.

Methodology:

Nakamoto Coefficient Calculation The Nakamoto Coefficient represents the minimum number of independent entities required to compromise a blockchain network (e.g. achieve 51% control of the network consensus). A higher coefficient implies higher decentralization.

1. **List Subsystem Providers:** Identify all entities (e.g. mining pools or validators) and their respective percentage of network control.
2. **Sort Descending:** Arrange the entities in descending order of their control percentages.
3. **Cumulative Sum:** Iteratively add the percentages from the top of the sorted list.

4. **Determine Threshold:** Stop when the cumulative sum exceeds 50% (i.e. $\geq 51\%$ or $> 50\%$ depending on protocol strictness).

5. **Result:** The Nakamoto Coefficient is the number of entities added to reach the threshold.

Worked Example:

Measuring Application Network Decentralization **Problem Statement:** A blockchain supporting a decentralized application uses Proof-of-Work. The hash power is distributed among 8 major mining pools as follows: Pool A (25%), Pool B (15%), Pool C (12%), Pool D (10%), Pool E (9%), Pool F (8%), Pool G (11%), Pool H (10%). Calculate the Nakamoto Coefficient for this blockchain network.
Solution:

1. **List and Sort Entities (Descending Order):**

Entity	Hash Power (%)
Pool A	25%
Pool B	15%
Pool C	12%
Pool G	11%
Pool D	10%
Pool H	10%
Pool E	9%
Pool F	8%

2. **Calculate Cumulative Sums:**

   - Step 1: Pool A = 25% (Cumulative: 25%)
   - Step 2: Pool A + Pool B = 25% + 15% = 40%
   - Step 3: Pool A + Pool B + Pool C = 40% + 12% = 52%

3. **Check Threshold:** The threshold is $> 50\%$. At Step 3, the cumulative hash power is 52%, which exceeds 50%.

4. **Determine Coefficient:** It took exactly 3 entities (Pool A, Pool B, and Pool C) to reach majority control. Therefore, the **Nakamoto Coefficient is 3**.

Methodology:

Decentralized Storage Distribution Algorithm For a DApp to be fully decentralized, its data (images, UI files) must not reside on a central server. Systems like IPFS use content addressing and chunking.

1. **File Chunking:** The file is broken down into fixed-size blocks (e.g. 256 KB).

2. **Hashing:** Each chunk is hashed using a cryptographic function (e.g. SHA-256) to produce a unique Content Identifier (CID).

3. **Merkle DAG Construction:** A Directed Acyclic Graph (DAG) is constructed where leaf nodes are the file chunks, and parent nodes contain the CIDs of their children. The root CID uniquely represents the entire file.

4. **Network Distribution:** Chunks are distributed across decentralized network nodes. When requested, the DApp uses the root CID to retrieve and reassemble the chunks from peers.

Worked Example:

Calculating File Chunks for Decentralized Storage **Problem Statement:** A decentralized application needs to store a dataset of exactly 1.5 MB on IPFS. Assume IPFS divides files into blocks of exactly 256 KB. Calculate the total number of chunks generated and the maximum number of direct edge links the root Merkle DAG node will have if no intermediate nodes are used.

Solution:

1. **Identify Inputs:** Total File Size = 1.5 MB
   Block Size = 256 KB
2. **Convert Units:** 1.5 MB = 1.5×1024 KB = 1536 KB
3. **Calculate Number of Chunks:**

$$\text{Chunks} = \frac{\text{Total Size}}{\text{Block Size}} = \frac{1536}{256}$$

$$\text{Chunks} = 6$$

4. **Determine Root Node Links:** Since the file is broken into 6 equal chunks and there are no intermediate nodes, the root Merkle DAG node will directly point to the CIDs of these 6 chunks. Thus, the root node will contain exactly **6 direct edge links**.

Appendix: Key Formulae

This appendix provides a quick reference to the key formulae and mathematical relationships discussed in this book, organized by unit.

Unit 1: Cryptography Basics

- **RSA Modulus**: The modulus n used in both public and private keys is the product of two prime numbers p and q.

$$n = p \times q$$

- **Euler's Totient Function**: Determines the number of positive integers less than or equal to n that are relatively prime to n. For primes p and q:

$$\phi(n) = (p-1) \times (q-1)$$

- **Simple Hash Function**: A basic hashing operation summing ASCII values modulo m.

$$H(x) = \left(\sum \text{ASCII}(x)\right) \pmod{m}$$

Unit 2: Distributed Systems & Consensus

- **RSA Encryption**: Computes the ciphertext C using the message M, public exponent e, and modulus n.

$$C = M^e \pmod{n}$$

- **RSA Decryption**: Recovers the message M using the ciphertext C, private exponent d, and modulus n.

$$M = C^d \pmod{n}$$

- **Gossip Protocol Infection Spread**: The number of infected nodes I_r at round r, given the fanout factor k.

$$I_r = I_{r-1} \times (1+k) = (1+k)^r$$

- **Gossip Protocol Rounds**: The total number of rounds R required to infect all N nodes.

$$R = \lceil \log_{1+k}(N) \rceil$$

- **Byzantine Fault Tolerance (BFT)**: The minimum number of total nodes N required to tolerate f faulty nodes.

$$N \geq 3f + 1$$

- **BFT Quorum Size**: The minimum number of nodes Q required to reach consensus.

$$Q = 2f + 1$$

Unit 3: Blockchain Mechanics & Security

- **Maximum Faulty Nodes**: The maximum number of malicious nodes f a network of size N can tolerate in BFT.
$$f = \lfloor \frac{N-1}{3} \rfloor$$
- **Difficulty and Target**: The relationship between network difficulty D, maximum target T_{max}, and the current target T.
$$D = \frac{T_{max}}{T} \implies T = \frac{T_{max}}{D}$$
- **Block Success Probability**: The probability p of a single hash successfully meeting the target T, where N_{bits} is the hash length.
$$p = \frac{T}{2^{N_{bits}}}$$
- **Expected Hashes per Block**: The average number of hashes H_{exp} required to find a valid block.
$$H_{exp} = \frac{1}{p} = \frac{2^{N_{bits}}}{T}$$
- **Halving Epochs**: The number of halving events n that have occurred by block height h, where I is the halving interval.
$$n = \lfloor \frac{h}{I} \rfloor$$
- **Block Reward**: The block reward R_h at halving epoch n, given the initial reward R_0.
$$R_h = \frac{R_0}{2^n}$$
- **51% Attack Condition**: A successful majority attack occurs when the attacker's hash rate H_{att} exceeds the network's hash rate H_{net}.
$$H_{att} > H_{net}$$
- **Attacker Catch-up Probability**: The probability P_z that an attacker catches up from z blocks behind, where q is the attacker's block generation probability and p is the honest network's probability.
$$P_z = \left(\frac{q}{p}\right)^z$$
- **Cost of Attack**: The cost to sustain a 51% attack, calculated hourly or totally, given the cost C per hash unit.
$$C_{hourly} = H_{att} \times C$$
$$C_{total} = C_{hourly} \times \text{Duration}$$

Unit 4: Smart Contracts & Ethereum Basics

- **Unit Conversion**: Relation between Gwei and Ether.
$$1 \text{ Gwei} = 10^{-9} \text{ ETH}$$
- **Transaction Fee**: The fee required to execute a transaction on Ethereum.
$$\text{Transaction Fee} = \text{Gas Used} \times \text{Gas Price}$$

Unit 5: Storage & Advanced Concepts

- **Transaction Fee (Algebraic)**: Using variables for Fee F, Gas Used G_u, and Gas Price P_g.

$$F = G_u \times P_g$$

- **Data Chunking**: Calculating the number of chunks required to split data for storage.

$$\text{Chunks} = \left\lceil \frac{\text{Total Size}}{\text{Block Size}} \right\rceil$$