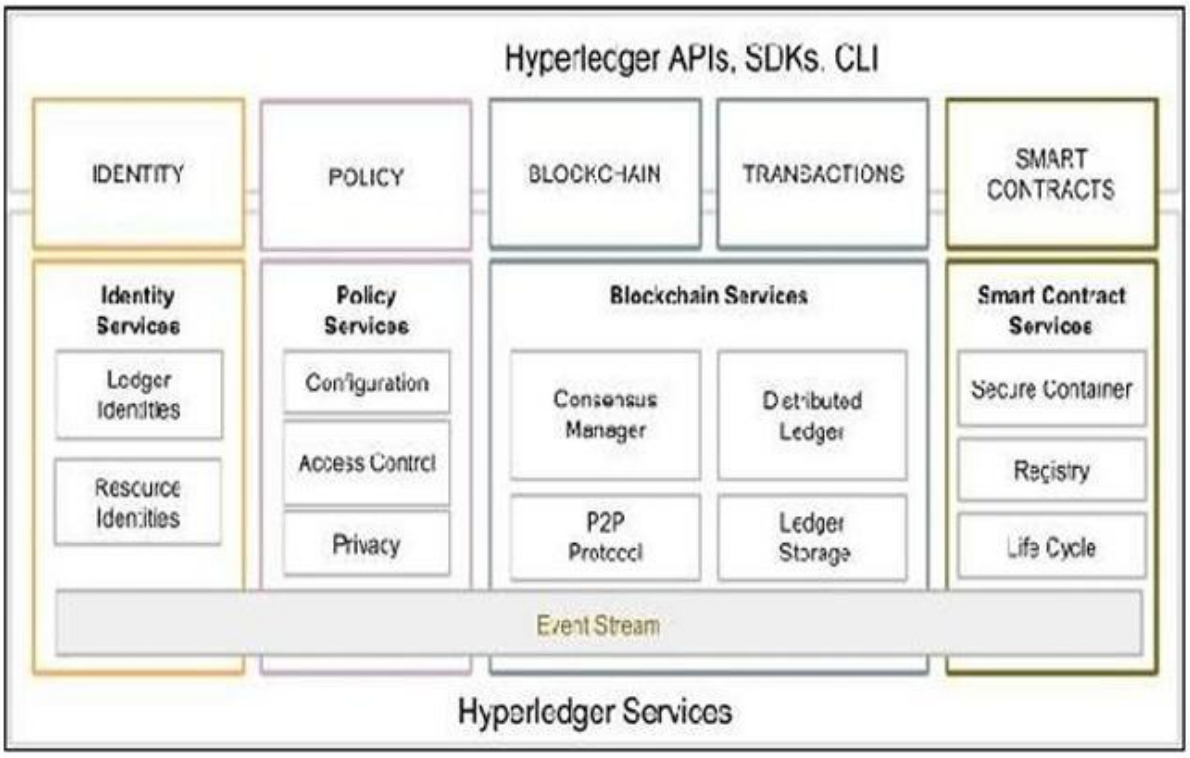
Module 5

1. Give the architecture of Hyperledger as a protocol and explain.



Architecture of Hyperledger as a Protocol

Hyperledger is not a single blockchain, but a project hosted by the Linux Foundation to develop open-source blockchain frameworks and tools for business use. As a protocol, Hyperledger defines a reference architecture to guide the development of permissioned distributed ledger platforms.

Reference Architecture Components

The Hyperledger reference architecture has two main parts:

1. Hyperledger Services
2. Hyperledger APIs, SDKs, and CLI

(i) Hyperledger Services:

- Provide essential blockchain functionalities such as:
    - Identity services – for managing participant identities and access.
    - Policy services – for defining and enforcing business rules.
    - Blockchain services – for ledger management, consensus, and data storage.
    - Smart contract services – for running business logic as chaincode.

- Services communicate using an **event stream** (via gRPC channels), enabling event-driven responses from applications. Events can be predefined or custom and emitted by validating peers or chaincode.

(ii) Hyperledger APIs, SDKs, and CLI:

- Offer interfaces to interact with blockchain services.
- Enable application development in multiple languages via SDKs.
- Command-line interfaces allow testing and managing blockchain networks.

Key Requirements of Hyperledger Protocol Architecture

Hyperledger's architecture is designed to meet specific enterprise needs:

✅ **Modular Approach:**

- Supports "plug-and-play" modules.
- Allows easy customization of storage, consensus, access control, and cryptography.
- Businesses can swap modules to meet specific needs.

✅ **Privacy and Confidentiality:**

- Emphasizes strong cryptographic support.
- Lets users choose cryptographic modules based on their security needs.
- Supports private transactions and contracts.

✅ **Identity Management:**

- Uses flexible **Public Key Infrastructure (PKI)**.
- Allows granular access control and optional anonymity.
- Membership services manage user registration and permissions.

✅ **Auditability:**

- Maintains an immutable audit trail of all identities and activities on the ledger.
- Supports regulatory compliance requirements.

✅ **Interoperability:**

- Aims to enable communication between different blockchain networks.
- Envisions standard protocols to allow exchange of information across ledgers.

✅ **Portability:**

- Designed to run across various platforms without code changes.

- Ensures uniform development and deployment in diverse environments.

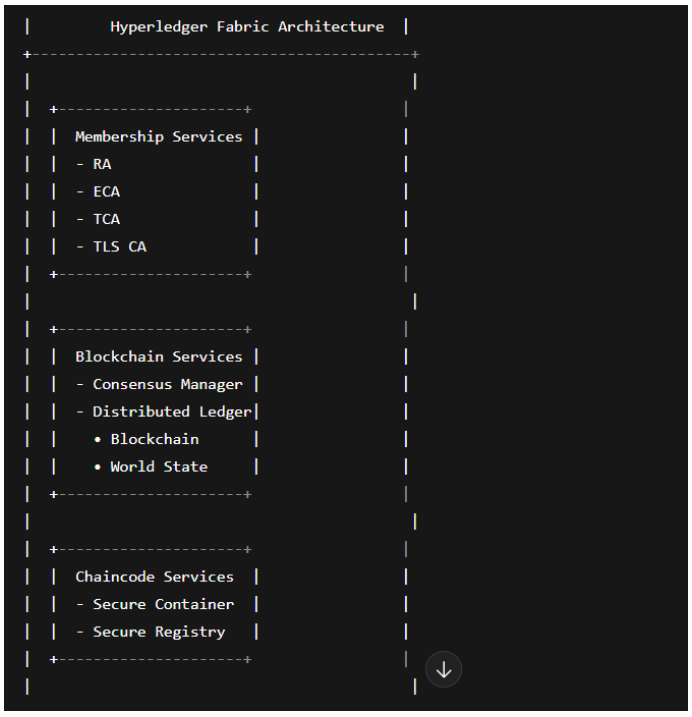
Summary of the Protocol Architecture

Hyperledger's reference architecture acts as a **blueprint** for building permissioned blockchain platforms. It emphasizes:

- **Flexibility** via modular design.
- **Security** through strong cryptography and managed identities.
- **Interoperability** with other blockchain systems.
- **Scalability** by allowing businesses to choose the best modules for their needs.

This architecture enables organizations to build blockchain networks tailored to their industry requirements while maintaining high standards of security, performance, and compliance.

2. Discuss about the organization and categories of Hyperledger fabric architecture.



Organization and Categories of Hyperledger Fabric Architecture

Hyperledger Fabric is one of the main projects under the Hyperledger umbrella. It is designed to provide a modular, flexible, and secure foundation for developing permissioned blockchain networks.

Fabric architecture is logically organized into three main categories based on the type of service provided:

---

1 Membership Services

- Purpose: Provides access control and identity management.
- Functions:
    - User identity validation.
    - User registration.
    - Assigning roles and permissions.
- Components:
    - Registration Authority (RA): Authenticates users and issues registration credentials.
    - Enrollment Certificate Authority (ECA): Issues long-term certificates (Ecerts) for identification.
    - Transaction Certificate Authority (TCA): Issues transaction certificates derived from Ecerts for signing transactions.
    - TLS Certificate Authority: Issues TLS certificates to secure network communications.
- Key Feature: Uses Public Key Infrastructure (PKI) to manage identities securely.

---

2 Blockchain Services

- Purpose: Manages the core ledger and consensus mechanisms.
- Components:
- Consensus manager Consensus manager is responsible for providing the interface to the consensus algorithm. This serves as an adapter that receives the transaction from other Hyperledger entities and executes them under criteria according to the type of algorithm chosen. Consensus is pluggable and currently there are three types of consensus algorithm available in Fabric, namely the batch PBFT protocol, SIEVE algorithm, and NOOPS.
- Distributed ledger Blockchain and world state are two main elements of the distributed ledger. Blockchain is simply a linked list of blocks (as introduced in earlier chapters) and world ledger is a key-value database. This database is used by smart contracts to store relevant states during execution by the transactions. The blockchain consists of blocks that contain transactions. These transactions contain chaincode, which runs transactions that can result in updating the world state. Each node saves the world state on disk in RocksDB. The following diagram shows a typical block in the Hyperledger Fabric with the relevant fields
- Block Structure Fields:
    - Version
    - Timestamp
    - Transaction Hash (Merkle root)

    - State Hash (Merkle root of world state)
    - Previous Hash
    - Consensus Metadata
    - Non-Hash Data
- Communication:
    - Built on **gRPC** protocol buffers.
    - Message types: Discovery, Transaction, Synchronization, and Consensus.

---

3 Chaincode Services

- **Purpose:** Supports execution of smart contracts (called *chaincode* in Fabric).
- Components:
    - Secure Container:
        - Chaincode runs in Docker containers.
        - Supports languages like Go, Java, Node.js.
    - Secure Registry:
        - Maintains records of chaincode images.
- Features:
    - Isolates execution for security.
    - Allows chaincode to be written in any mainstream language if supported.

Additional Aspects of the Architecture

✅ Events System:

- Validator nodes and chaincode can emit events.
- External applications can subscribe and react to these events via adapters.

✅ APIs and CLIs:

- REST APIs for interaction with the ledger.
- Command-line interfaces for management and testing.

✅ Peers (Nodes):

- **Validating Peers:** Run consensus, validate transactions, update the ledger.
- **Non-Validating Peers:** Relay transactions, provide verification and REST services, maintain user certificates.

✅ Applications:

- Typically built with a frontend (JavaScript/HTML) that communicates with backend chaincode via APIs.

Summary

The **organization of Hyperledger Fabric architecture** ensures:

- **Membership services** for secure identity and access control.
- **Blockchain services** for ledger management and consensus.
- **Chaincode services** for smart contract execution in secure environments.

This modular design allows businesses to tailor blockchain networks to their specific needs, ensuring **security, scalability, privacy, and flexibility.**

3. Define Sawtooth lake. Explain the novel concepts of Sawtooth lake.

✅ Definition of Sawtooth Lake

Sawtooth Lake is a blockchain platform project proposed by **Intel** in April 2016 under the Hyperledger umbrella.
It is designed as a **flexible, modular, and enterprise-grade distributed ledger** that supports both permissioned and permissionless setups.
Sawtooth Lake's goal is to **simplify blockchain application development** by providing innovative concepts that improve security, scalability, and flexibility across business use cases.

✅ Novel Concepts of Sawtooth Lake

Sawtooth Lake introduces **two main novel concepts**:

---

1 Proof of Elapsed Time (PoET) Consensus Algorithm

- PoET is a **novel consensus mechanism** developed by Intel.
- It uses a **Trusted Execution Environment (TEE)**, specifically **Intel's SGX** (Software Guard Extensions), to ensure randomness and security.

Key features of PoET:

- ✅ Random leader election without expensive computations (unlike Bitcoin's Proof of Work).
- ✅ Nodes wait for a randomly chosen time period before proposing a block.
- ✅ Ensures fairness and energy efficiency by avoiding high energy usage.
- ✅ Trusted execution guarantees that waiting times are honest and tamper-proof.

Advantages over traditional Proof of Work:

- Reduces energy consumption drastically.
- Provides a secure and fair leader election without mining races.
- Supports both **permissioned** (with known validators) and **permissionless** networks.

---

2 Transaction Families

- Sawtooth Lake **decouples business logic from the core ledger** using "transaction families."

- Instead of one monolithic smart contract layer, business logic is **organized into domain-specific transaction families**.

  Key features of Transaction Families:

  ✅ Define **rules and structures** for specific domains (e.g., supply chain, finance).

  ✅ Separate transaction processing logic from the ledger itself.

  ✅ Allow flexible, secure, and clear design of business rules.

  ✅ Reduce security risks by limiting the scope of smart contract logic to only what is needed for that domain.

  Advantages over generic smart contract models (like Ethereum's EVM):

- Reduces attack surface and vulnerabilities.
- Avoids unnecessary complexity by supporting only relevant operations.
- Easier to audit and maintain.

Examples of Transaction Families in Sawtooth Lake:

✅ **Endpoint Registry:** For registering ledger services.

✅ **IntegerKey:** For testing deployed ledgers.

✅ **MarketPlace:** For buying, selling, and trading operations.

Intel even demonstrated **Sawtooth Bond** as a **proof of concept** for a bond trading platform built with transaction families.

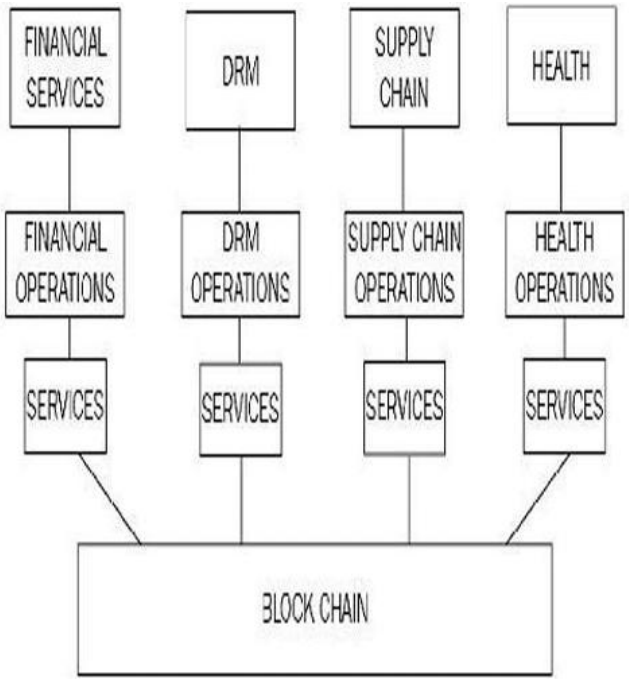
✅ **Conclusion**

Sawtooth Lake stands out among blockchain platforms because of its **innovative concepts**:

✅ **PoET Consensus:** Ensures fair, low-energy, secure leader election using Intel SGX.

✅ **Transaction Families:** Offer modular, domain-specific, secure smart contract design.

These innovations make Sawtooth Lake suitable for **enterprise applications** that need flexibility, scalability, security, and efficiency.



4. What is Corda. List and explain the components of the Corda

✅ What is Corda?

Corda is an open-source distributed ledger platform developed by **R3**, designed specifically for the **financial industry** and other regulated sectors.

Unlike traditional blockchains (like Bitcoin or Ethereum), **Corda is *not* a blockchain in the typical sense.**

✅ It does **not** group transactions into blocks or broadcast them globally.

✅ Instead, Corda provides **direct, private transactions** between only those parties who need to see them.

✅ It ensures **consensus over transaction validity and uniqueness** without compromising privacy.

Corda is designed to solve real-world business problems such as:

- Reducing reconciliation costs between organizations.
- Ensuring regulatory compliance.
- Enabling legally-enforceable smart contracts with high privacy.

✅ Components of Corda

Below are the main components of the Corda architecture, explained in detail:

---

1. State Objects

- Smallest unit of data in Corda, representing an **agreement or asset** shared between parties.
- States are **immutable**: they can only be created or consumed by transactions.
- Contain references to **contract code** (which validates changes) and optional **legal prose** (the human-readable legal agreement).
- Acts like a **state machine**: transitions from one state to another through transactions.
  ✅ *Example:* A loan agreement between two banks.

---

2. Contract Code

- Defines **business rules** and **validation logic** for state transitions.
- Ensures that all changes to states via transactions are **valid** and comply with predefined rules.
- Written in JVM languages like **Kotlin** or **Java**.
  ✅ *Example:* A contract that enforces payment terms between parties.

3.Legal Prose

- **Optional** human-readable text attached to the contract code.

- Represents the **legally-binding terms** of the agreement.
- Ensures that the **smart contract aligns with traditional legal contracts**.
  ✅ Bridges the gap between code and legal enforceability.

---

4. Transactions

- Define how **state objects change** from inputs to outputs.
- Are **digitally signed** by all required parties to ensure authenticity.
- Transactions in Corda are **not broadcast globally**—only shared with parties involved.

Key Elements of a Transaction:

✅ **Input References:** States consumed.
✅ **Output States:** New states created.
✅ **Attachments:** Hashes of attached zip files containing code or documents.
✅ **Commands:** Define operations, including required signers.
✅ **Signatures:** Ensure transaction integrity and authenticity.
✅ **Type:** Normal or Notary-changing.
✅ **Timestamp:** Defines the time window during which the transaction is valid.
✅ **Summaries:** Text description of the transaction.

5.Consensus

- Corda's consensus has **two goals**:
  ✅ **Validity:** Ensuring all business rules (contract code) are correctly followed.
  ✅ **Uniqueness:** Ensuring no double spending.
- Consensus is achieved without mining or proof-of-work.
- Relies on **Notary Services** that sign transactions if they are valid and unique.
  ✅ Notaries can use consensus protocols like **PBFT** or **Raft**.

---

6 Flows

- Define **automated workflows** between nodes.
- Allow nodes to **communicate securely and agree on transaction details**.
- Written as **asynchronous state machines** that can suspend and resume.
  ✅ *Example:* A payment flow where two banks exchange transaction details and signatures.

---

7. Nodes

- Each organization runs its own **Corda node**.

- Nodes communicate **peer-to-peer** over secure channels.
  - ✅ Use **AMQP** protocol over **TLS** for security.
  - ✅ Messages are encoded in binary for efficiency.
  - ✅ Each node maintains a **local relational database** for storing states.
  - ✅ Types of nodes: Regular node, Notary, Oracle, Network Map Service.

---

8. Vaults

- Store **state objects** relevant to that node.
- Like **wallets** in Bitcoin but more advanced.
  - ✅ Contains on-ledger and off-ledger data.
  - ✅ Uses standard relational databases (like H2) for storage.
  - ✅ Supports SQL queries for easy integration with existing systems.

---

9. Notary Service

- Prevents **double-spending** by ensuring transaction uniqueness.
- Signs transactions that are valid and not already consumed.
  - ✅ Can be deployed in clusters for **load balancing**.
  - ✅ Supports consensus algorithms like **BFT** or **Raft**.
  - ✅ Recommended to be geographically close to participants to reduce latency.

10.Oracle Service

- Provides **external data** to the ledger.
  - ✅ Signs transactions containing **verified facts**.
  - ✅ Can serve as **data providers** or **data verifiers**.
  - ✅ *Example:* An Oracle signing interest rate data used in a loan contract.

---

11. Network Map Service

- Maintains a **directory of all nodes** on the network.
- Maps node identities to network addresses.
  - ✅ Ensures **secure discovery and communication** between nodes.
  - ✅ Nodes register themselves on the network map when they start up.

---

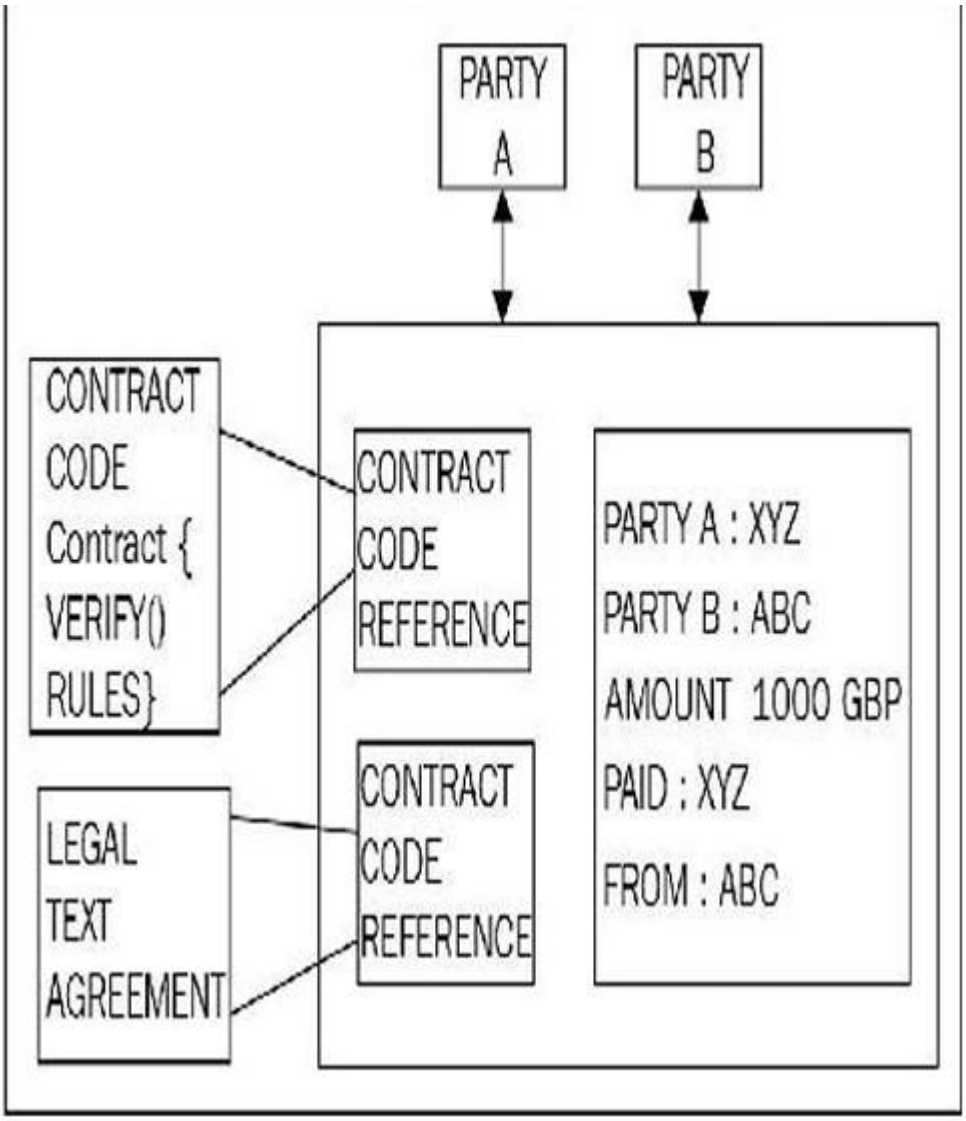
12. Corda apps (Corda Distributed Apps)

- Complete **applications** built on top of Corda.
  - ✅ Include state objects, contract code, flows, APIs, and UI components.

- ✅ Typically written in **Kotlin** or **Java**.
- ✅ Can be deployed to any Corda node.
- ✅ Designed to solve real-world use cases like loan processing, trade finance, KYC, etc.

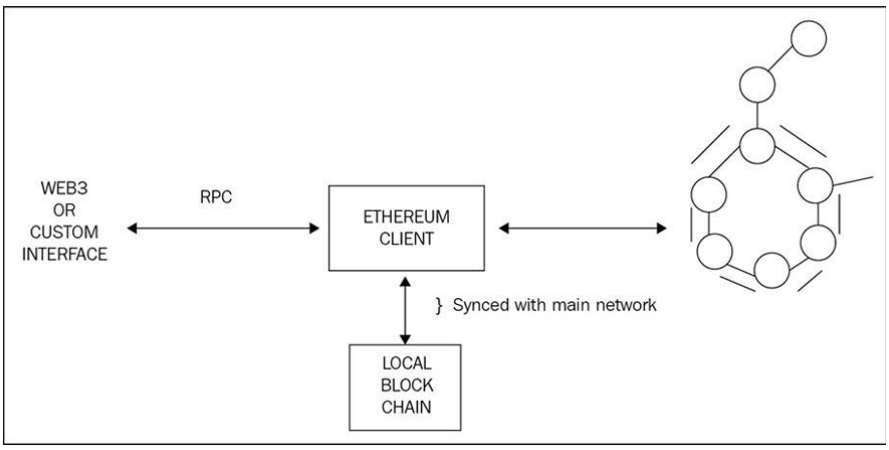
5. Explain corda architecture

Write the same answer as above but learn this diagram below:



Module 4

1. Explain the Ethereum Stack with a suitable diagram.



Explain the Ethereum Stack with a suitable diagram

The **Ethereum stack** is a comprehensive suite of tools, libraries, and services that together enable the development, deployment, and interaction with decentralized applications (dApps) on the Ethereum blockchain. Each layer plays a specific role in facilitating smooth and secure operations on the Ethereum network.

```sql
+-----------------------------------------------------------+
|                  Decentralized Applications (dApps)       |
+-----------------------------------------------------------+
|     MetaMask / Infura / Web Interfaces (UI Layer)         |
+-----------------------------------------------------------+
|   Truffle       |     Ganache      |     Web3.js          |
| (Framework)     | (Local testnet)  | (JS library)         |
+-----------------------------------------------------------+
|       Ethereum Client (e.g., geth, parity)                |
+-----------------------------------------------------------+
|     Ethereum Virtual Machine (EVM)                        |
+-----------------------------------------------------------+
|             Ethereum Blockchain (PoS)                     |
+-----------------------------------------------------------+
```

Components of the Ethereum Stack

1. **Ethereum Blockchain (PoS)**
    - The base layer that stores all transactions and smart contracts.
    - Ethereum now uses **Proof of Stake (PoS)** consensus.

2. Ethereum Virtual Machine (EVM)
    - A runtime environment that executes smart contracts.
    - Every node on Ethereum runs the EVM.
3. Ethereum Clients
    - Examples: geth (Go Ethereum), parity (Rust).
    - Clients sync with the Ethereum network, mine blocks, and manage accounts.
4. Web3.js
    - A JavaScript library that allows interaction between frontend applications and Ethereum.
    - Connects to Ethereum nodes via RPC (Remote Procedure Call).
5. Truffle
    - A development framework for Ethereum.
    - Helps in compiling, testing, and deploying smart contracts.
6. Ganache
    - A personal local Ethereum blockchain.
    - Used for testing smart contracts before deploying them on the mainnet.
7. MetaMask & Infura
    - MetaMask: A browser extension for managing Ethereum accounts and signing transactions.
    - Infura: Provides remote access to Ethereum nodes (useful if the developer doesn't want to run a full node).

Smart Contracts & Solidity

- Smart contracts are self-executing scripts written in Solidity, a high-level programming language designed for Ethereum.
- Once deployed, they run on the EVM and enforce contract rules automatically.

Conclusion

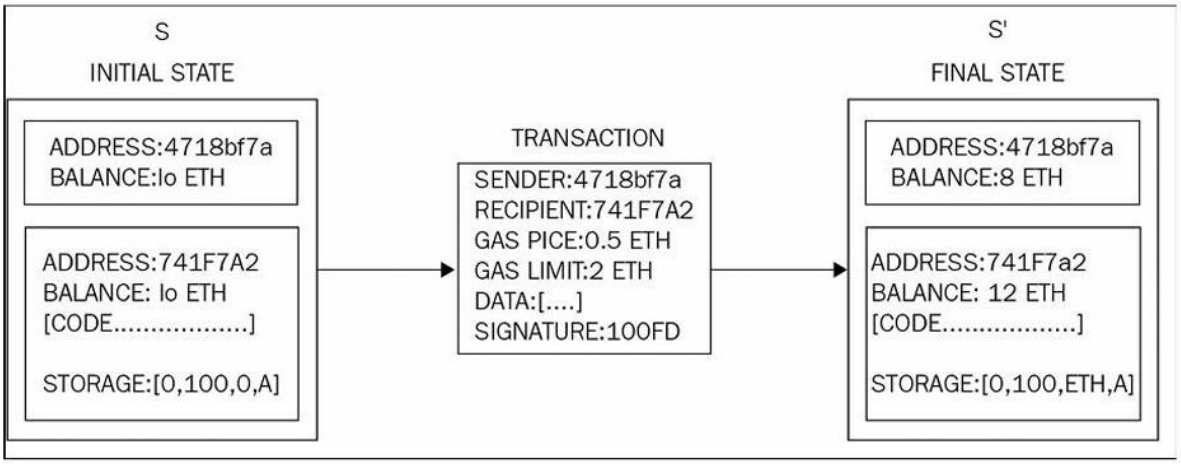
The Ethereum stack forms the backbone of dApp development. From the blockchain base layer and EVM, up through clients like geth and tools like Truffle and Web3.js, it enables decentralized, secure, and flexible application development. Understanding each component's role helps developers effectively build and interact with the Ethereum ecosystem.

2. What is Ethereum Blockchain? Discuss with a state transition diagram.

Ethereum is a decentralized, open-source blockchain platform designed to enable the development and execution of smart contracts and decentralized applications (dApps) without any downtime, fraud, or interference from third parties.

The Ethereum blockchain can be **viewed as a state machine** that transitions from one state to another through transaction executions.

- The **genesis state** is the starting point.
- Each transaction changes the global state by transferring value or executing smart contract code.
- This process is deterministic: all nodes compute the same result and reach **consensus**.



Ethereum, just like any other blockchain, can be visualized as a transaction- based state machine. This is mentioned in the Ethereum yellow paper written by Dr. Gavin Wood.

• The idea is that a genesis state is transformed into a final state by executing transactions incrementally. The final transformation is then accepted as the absolute undisputed version of the state.

• In the following diagram, the Ethereum state transition function is shown, where a transaction execution has resulted in a state transition.

• In the preceding example, a transfer of 2 Ether from Address 4718bf7a to Address 741f7a2 is initiated. The initial state represents the state before the transaction execution and the final state is what the morphed state looks like

State Transition Function

Ethereum uses a **state transition function** to apply transactions to the current state and produce a new state:

[STATE_n+1 = ƒ(STATE_n, TX)]

Where:

- STATE_n = current state

- TX = transaction
- ƒ = state transition function

3. Give the different types of Transactions in Ethereum and also explain the fields included in these transactions

1 Message Call Transactions

- Transfers value or calls existing smart contracts.
- Used to send Ether or invoke contract functions.
- Does not create new contracts.
- Represents a call or message sent to an existing account (EOA or contract).
- Executes code if the destination is a contract account.
- Causes state transitions by transferring value or executing contract logic.

✅ *Example:* Sending 5 ETH to another user or calling a function in a deployed smart contract.

2. Contract Creation Transactions

- Used to deploy new smart contracts on the blockchain.
- Includes contract initialization code (EVM bytecode).
- Creates a new contract account with associated code.
- Deploys a new smart contract to the Ethereum blockchain.
- Contains the contract's initialization (constructor) code.
- On success, a new contract account is created with its own address.

✅ *Example:* Deploying a new ERC20 token contract.

Fields included in Ethereum transactions:

• Nonce- Nonce is a number that is incremented by one every time a transaction is sent by the sender. It must be equal to the number of transactions sent and is used as a unique identifier for the transaction. A nonce value can only be used once.

• gasPrice- The gasPrice field represents the amount of Wei required in order to execute the transaction.

• gasLimit- The gasLimit field contains the value that represents the maximum amount of gas that can be consumed in order to execute the transaction

• Value- Value represents the total number of Wei to be transferred to the recipient; in the case of a contract account, this represents the balance that the contract will hold.

• Signature- Signature is composed of three fields, namely v, r, and s. These values represent the digital signature (R, S) and some information that can be used to recover the public key (V).

• init/data - Byte array with input data for the message call (data) or contract initialization code (init)

• To - Recipient's address. For message calls, it's the destination account; for contract creation, it's empty.

Signing Process

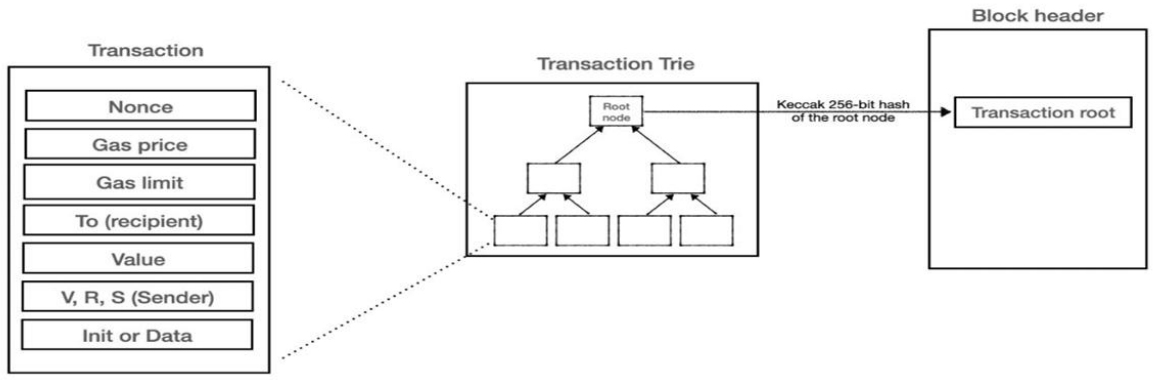
To sign a transaction, Ethereum uses the ECDSA signing function:

ECDSASIGN(Message, Private Key) = (V, R, S)

This produces the signature components (V, R, S) that validate the transaction's authenticity and allow the network to verify the sender's identity.

Transaction Inclusion in the Blockchain

Transactions are organized into a transaction trie, a modified Merkle-Patricia tree structure. The root hash of this trie is included in the block header after being hashed with the Keccak 256-bit algorithm. This structure enables efficient verification and secure inclusion of all transactions within a block, and is depicted below.



4. Briefly discuss about the operation of Ethereum Virtual Machine (EVM) with the help of a neat diagram.

The **Ethereum Virtual Machine (EVM)** is the decentralized, Turing-complete runtime environment on Ethereum nodes that executes smart contract code in a secure and deterministic way.

- Every Ethereum node runs the EVM to maintain consensus.
- EVM ensures that smart contract execution is **sandboxed**, **isolated**, and **predictable**.

✅ **Key Characteristics of EVM (From Notes)**

✅ **Stack-based Architecture**

- Operates as a **256-bit word stack machine**.
- Stack size limited to 1024 elements (Last In First Out—LIFO).

✅ **Deterministic Execution**

- Same input = same output on every node.
- Guarantees consensus.

✅ **Turing-Complete**

- Can perform any computation, given enough gas.
- Prevents infinite loops using **gas**.

✅ **Sandboxed Environment**

- No direct access to external resources (e.g., file system, network).
- Prevents external interference.

✅ **Gas System**

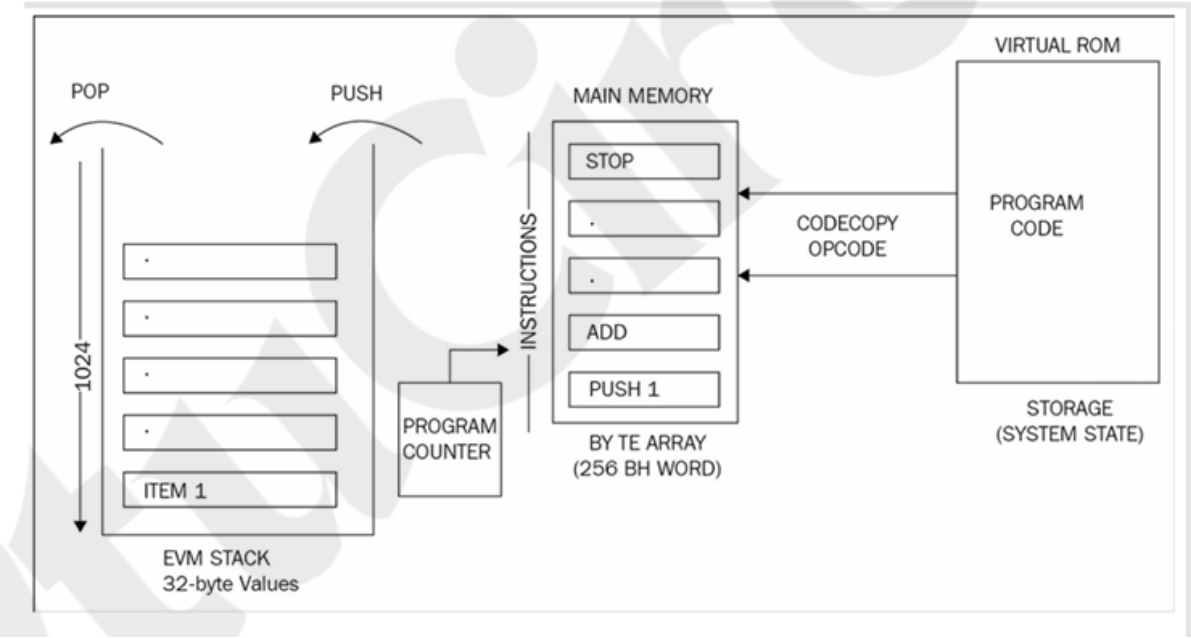
- Each operation has a predefined gas cost.
- Prevents denial-of-service (DoS) by limiting computational resources.

✅ **Supports Two Storage Types:**

- **Memory:** Temporary, cleared after execution (like RAM).
- **Storage:** Persistent key-value store on blockchain.

✅ **Big-Endian Word Size:**

- 256-bit wide words for cryptographic operations.



The **Ethereum Virtual Machine (EVM)** is the execution environment for smart contracts on Ethereum. It ensures **deterministic**, **sandboxed**, and **secure** execution of contract code across all nodes.

Here is the **step-by-step operation**:

---

1 Bytecode Input

- Smart contracts are written in high-level languages (e.g., Solidity) and compiled into **EVM bytecode**.
- This bytecode is the **program code** stored on-chain.
- When called, this code is loaded into the EVM for execution.

✅ *Purpose*: Ensures contracts can be run identically on any node.

---

2 Program Counter (PC)

- Holds the address of the **next instruction** to execute in the bytecode.
- Incremented step-by-step as instructions are executed.

✅ *Purpose*: Tracks execution progress in the program code.

---

3 Stack Operations

- EVM uses a **256-bit word stack** (Last-In-First-Out).
- Operations include **PUSH**, **POP**, **DUP**, **SWAP**.
- Maximum stack size is **1024 elements**.
- Used for storing **operands** and **intermediate results**.

✅ *Purpose*: Fundamental for instruction execution (like a calculator's memory).

---

4 Memory Access

- EVM has a temporary **byte-array memory**.
- Cleared after execution finishes (non-persistent).
- Instructions can **READ** and **WRITE** to memory during execution.
- Used for function parameters, local variables, etc.

✅ *Purpose*: Supports dynamic computations during contract execution.

---

5 Storage Access

- Persistent **key-value store** associated with each contract account.
- Used to store contract **state variables** permanently on the blockchain.

- Changes to storage are part of blockchain state transitions.
- Only modifiable via contract code execution.

✅ *Purpose*: Enables contracts to maintain state across calls and transactions.

---

6 CODECOPY Operation

- Special instruction that **copies program code** from contract's **ROM** to **memory**.
- Allows the contract to access and manipulate its own code during execution.

✅ *Purpose*: Enables dynamic code execution and introspection.

---

7 Gas Accounting

- Every opcode/instruction consumes a specific **gas cost**.
- Total gas is deducted from the transaction's **gas limit**.
- If gas runs out → execution **halts** with an **out-of-gas exception** and **reverts** changes (but gas is still spent).

✅ *Purpose*: Prevents infinite loops, controls resource usage, prevents denial-of-service (DoS).

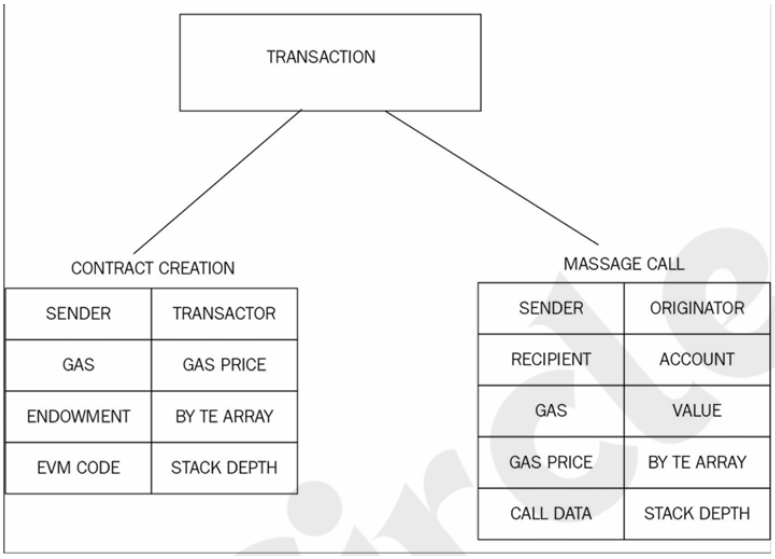
---

8 State Transition

- After successful execution:
    - Changes to **storage** are committed.
    - ETH transfers, contract calls, logs are finalized.
- If execution fails:
    - State changes **revert**, except **gas is consumed**.

✅ *Purpose*: Ensures deterministic, consistent blockchain update

5. What are Contract creation transaction and Message call transaction? Explain with a suitable diagram.



1 Message Call Transactions

- Transfers value or calls existing smart contracts.
- Used to send Ether or invoke contract functions.
- Does not create new contracts.
- Represents a call or message sent to an existing account (EOA or contract).
- Executes code if the destination is a contract account.
- Causes state transitions by transferring value or executing contract logic.

✅ *Example:* Sending 5 ETH to another user or calling a function in a deployed smart contract.

📌 Essential Parameters for Message Call

✔ Sender

- The account initiating the transaction.
- Signs the transaction.

✔ Transaction Originator

- The address at the root of the entire call chain.
- Remains the same across nested calls.

✔ Recipient

- Target account address:
    - Externally Owned Account (EOA) or
    - Contract Account whose code will be executed.

✔ Available Gas

- Gas limit set for the call.
- Ensures bounded execution.

✔ Gas Price

- Amount willing to pay per unit gas.
- Rewards miners.

✔ Input Data (Arbitrary Byte Array)

- Encoded function name and parameters (for smart contracts).
- For simple ETH transfers, usually empty.

✔ Current Depth

- Current level of nested calls in the EVM call stack.
- Used to prevent exceeding maximum allowed depth.

📌 Execution Outcome

- Always results in a **state transition** (e.g., balance changes, contract logic).
- Produces **output data** if it calls a function returning a value.
- Can succeed or revert if an error occurs.

✅ *Special Note:*

- Output data is often only used if **called internally** by another contract.

2. Contract Creation Transactions

- Used to deploy new smart contracts on the blockchain.
- Includes contract initialization code (EVM bytecode).
- Creates a new contract account with associated code.
- Deploys a new smart contract to the Ethereum blockchain.

- Contains the contract's initialization (constructor) code.
- On success, a new contract account is created with its own address.

✅ *Example:* Deploying a new ERC20 token contract.

Essential Parameters for Contract Creation

✔ Sender

- The externally owned account (EOA) initiating the transaction.

✔ Original Transactor

- Origin address that initiated the entire call chain.

✔ Available Gas

- Total gas allocated for executing the transaction.
- Ensures resources are bounded.

✔ Gas Price

- Price per unit of gas willing to be paid (in Wei).
- Incentivizes miners to include the transaction.

✔ Endowment

- Initial amount of **Ether** sent along with the contract.
- Sets the contract's balance upon creation.

✔ Initialization EVM Code

- A **byte array** containing the contract's constructor code.
- Executes once on creation to initialize contract storage and logic.

✔ Current Depth

- Current **call/creation stack depth** (number of nested calls).
- Ensures maximum stack depth is not exceeded (prevents recursion attacks).

📌 Address Generation for Contracts

- New contract addresses are **160-bit** in length.
- Defined as **rightmost 160 bits** of the Keccak hash of the **RLP encoding** of:

- Sender's address.
- Sender's transaction nonce.

✅ *Meaning*: Address is **predictable**, based on sender and nonce.

📌 **Account Initialization Process**

- **Nonce** initially set to zero.
- **Balance** initialized to the endowment value.
- **Storage** starts empty.
- **Code Hash** is initially the Keccak-256 hash of an empty string.
- EVM executes **Initialization Code** to set contract's logic.

📌 **Execution Outcome**

- If execution **runs out of gas** or fails → no state change, no contract created.
- If execution **succeeds** → contract account is created with its code and balance.
- **Gas cost** must be paid regardless of success.

✅ *Homestead Update:*

- Prevented contracts from being created if constructor fails (fixing earlier bugs).

6. Summarize the types of network which contribute to the consensus mechanism in ethereum.

The Ethereum network is a peer-to-peer network where nodes participate in order to maintain the blockchain and contribute to the consensus mechanism. Networks can be divided into three types, based on the requirements and usage. These types are described in the following subsections.

The mainnet

The mainnet is the current live network of Ethereum. Its network ID is 1 and its chain ID is also 1. The network and chain IDs are used to identify the network. A block explorer that shows detailed information about blocks and other relevant metrics is available at https://etherscan.io. This can be used to explore the Ethereum blockchain.

Testnets

There is a number of testnets available for Ethereum testing. The aim of these test blockchains is to provide a testing environment for smart contracts and DApps before being deployed to the production live blockchain. Moreover, being test networks, they also allow experimentation and research. The main testnet is called Ropsten, which contains all the features of other smaller and special-purpose testnets that were created for specific releases. For example, other testnets include

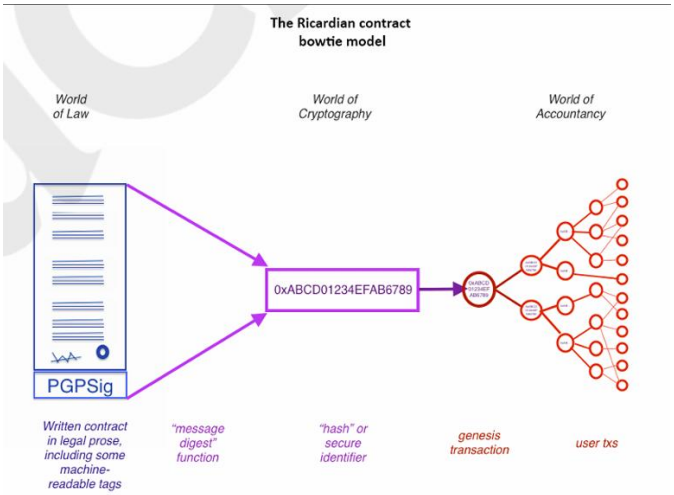
Kovan and Rinkeby, which were developed for testing Byzantium releases. The changes that were implemented on these smaller testnets have also been implemented in Ropsten. Now the Ropsten test network contains all properties of Kovan and Rinkeby.

Private nets

As the name suggests, these are private networks that can be created by generating a new genesis block. This is usually the case in private blockchain networks, where a private group of entities start their blockchain network and use it as a permissioned or consortium blockchain.

Module 3

1.Write a short note on Ricardian contract



• Ricardian contracts were initially used in a bond trading and payment system called Ricardo. The fundamental idea behind this contract is to write a document that is understood and accepted by both a court of law and computer software. Ricardian contracts address the challenge of the issuance of value over the internet. A Ricardian contract identifies the issuer and captures all the terms and clauses of the contract in a document to make it acceptable as a legally binding contract.

• A Ricardian contract is a document that has several of the following properties:

• It is a contract offered by an issuer to holders

• It is a valuable right held by holders and managed by the issuer

• It can be easily read by people (like a contract on paper)

• It can be read by programs (parsable, like a database)

• It is digitally signed

• It carries the keys and server information

• It is allied with a unique and secure identifier

The diagram shows a number of elements:

• The World of Law is on the left-hand side from where the document originates. This document is a written contract in legal prose with some machine-readable tags.

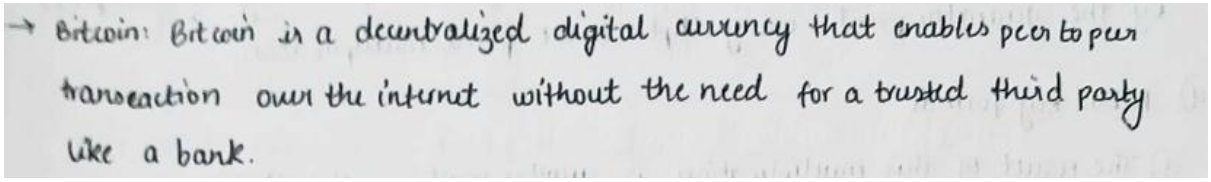
• This document is then hashed.

• The resultant message digest is used as an identifier throughout the World of Accountancy, as shown on the right-hand side of the diagram.

• The World of Accountancy element represents any accounting, trading, and information systems that are being used in the business to perform various business operations. The idea behind this flow is that the message digest generated by hashing the document is first used in a so-called genesis transaction, or first transaction, and then it is used in every transaction as an identifier throughout the operational execution of the contract. This way, a secure link is created between the original written contract and every transaction in the World of Accounting:

• A Ricardian contract is different from a smart contract in the sense that a smart contract does not include any contractual document and is focused purely on the execution of the contract. A Ricardian contract, on the other hand, is more concerned with the semantic richness and production of a document that contains contractual legal prose. The semantics of a contract can be divided into two types: operational semantics and denotational semantics

2. Define Bitcoin. Explain the generation of public key and private key in detail.



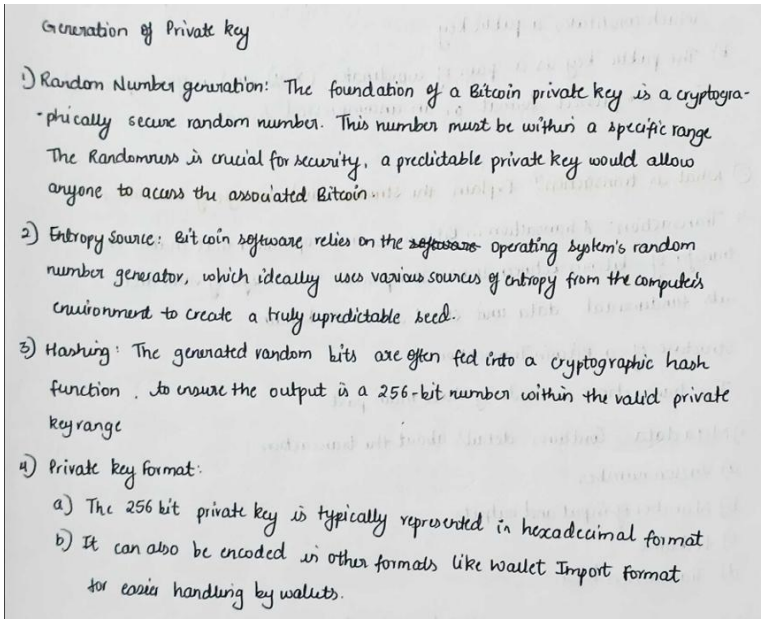
Bitcoin can be defined in multiple ways:

- **Bitcoin (capital B)** refers to the **Bitcoin protocol** — a combination of peer-to-peer (P2P) network, cryptographic protocols, and software that allows the transfer and management of the **digital currency** called bitcoin.
- **bitcoin (lowercase b)** refers to the **currency unit** itself used for transactions.
- It is a **decentralized digital currency** that eliminates the need for intermediaries like banks and solves the **double-spending problem**.

Bitcoin was introduced in the 2008 paper *"Bitcoin: A Peer-to-Peer Electronic Cash System"* by **Satoshi Nakamoto**.

Private Key Generation:

- A **private key** is simply a **random 256-bit number**.
- It is generated using secure cryptographic methods (via wallet software).
- The number must be within a valid range defined by the **secp256k1** elliptic curve used in Bitcoin.
- Example format (in hexadecimal): e8f32e723decf4051aefac8e6b43e3a48...



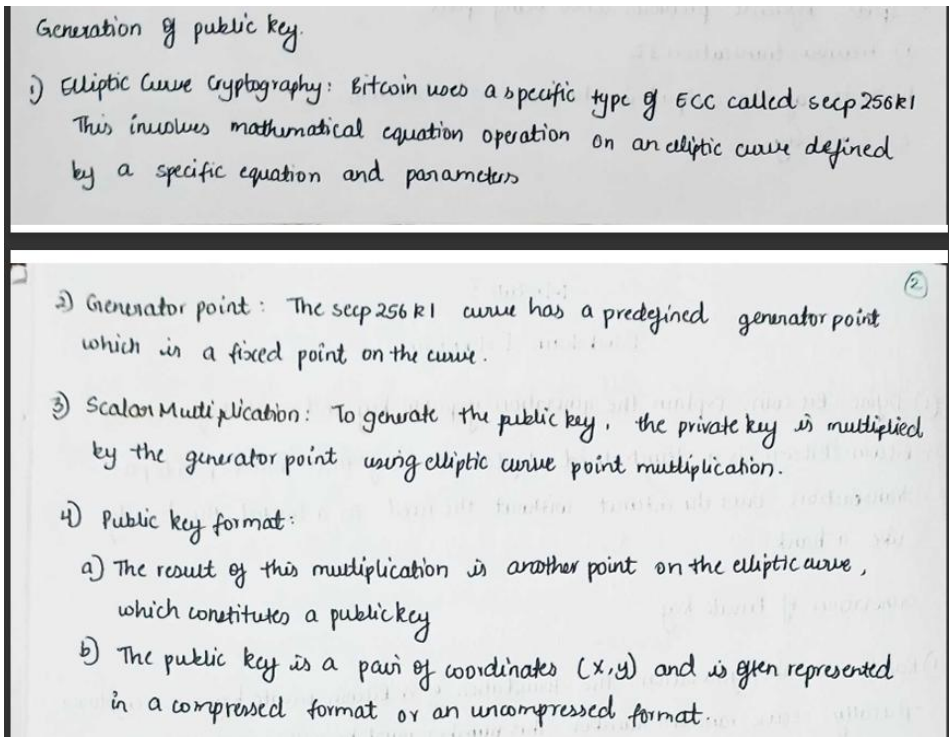
Generation of Private key

1) Random Number generation: The foundation of a Bitcoin private key is a cryptographically secure random number. This number must be within a specific range. The Randomness is crucial for security, a predictable private key would allow anyone to access the associated Bitcoin.

2) Entropy source: Bitcoin software relies on the ~~software~~ operating system's random number generator, which ideally uses various sources of entropy from the computer's environment to create a truly unpredictable seed.

3) Hashing: The generated random bits are often fed into a cryptographic hash function, to ensure the output is a 256-bit number within the valid private key range.

4) Private key format:
   a) The 256 bit private key is typically represented in hexadecimal format
   b) It can also be encoded in other formats like Wallet Import format for easier handling by wallets.



Generation of public key.

1) Elliptic Curve Cryptography: Bitcoin uses a specific type of ECC called secp256k1. This involves mathematical equation operation on an elliptic curve defined by a specific equation and parameters.

(2)

2) Generator point: The secp256k1 curve has a predefined generator point which is a fixed point on the curve.

3) Scalar Multiplication: To generate the public key, the private key is multiplied by the generator point using elliptic curve point multiplication.

4) Public key format:
   a) The result of this multiplication is another point on the elliptic curve, which constitutes a public key
   b) The public key is a pair of coordinates (x,y) and is often represented in a compressed format or an uncompressed format.

->. Bitcoin Address Creation (from Public Key):

- To create a **Bitcoin address** (which is shared to receive funds), the public key is **hashed** using:
  - SHA-256
  - Then RIPEMD-160
- The resulting value is the **Public Key Hash (PKH)**.
- This PKH is further encoded using Base58Check to form the final human-readable Bitcoin address (starts with 1 or 3).

3. What is a Transaction? Explain the structure and life cycle of the transactions.

✅ 1. What is a Transaction in Bitcoin?

A *transaction* is the fundamental operation in the Bitcoin system. It is the way bitcoins are transferred from one user to another.

Definition (as per your PDF):

- A transaction can be as simple as sending bitcoins to a Bitcoin address, or more complex with multiple inputs/outputs.
- Each transaction **consumes previous outputs (UTXOs)** and creates new outputs.
- Transactions are *not encrypted*, but *publicly visible* on the blockchain.
- Transactions solve the **double-spending problem** by ensuring that each coin is only spent once.

✅ 2. Structure of a Transaction (from "THE TRANSACTION DATA STRUCTURE" section)

A **Bitcoin transaction** consists of:

➜ (i) Metadata

- Includes values like:
  - Size of the transaction
  - Number of inputs and outputs
  - Transaction hash
  - Lock_time (indicates earliest time/height the transaction can be added to a block)
  - Version number

➜ (ii) Inputs

- Represents *coins being spent*.
- Each input refers to a previous output (UTXO) that is being consumed.
- Contains:
  - Reference to previous transaction's output
  - ScriptSig (Unlocking script): proves ownership by providing signature(s).

➜ (iii) Outputs

- Represents *new coins being created*.
- Contains:
    - Amount in satoshis
    - Locking script (ScriptPubKey): specifies conditions for spending (e.g., recipient's address/public key hash).

➜ (iv) Verification

- Performed using Bitcoin's scripting language.
- Scripts are simple and stack-based.
- Ensures conditions specified in outputs are satisfied.

➜ Standard Output Example

- Pay-to-Public-Key-Hash (P2PKH): the most common output type, locks the output to a hash of the recipient's public key.

✅ 3. Transaction Lifecycle (from "THE TRANSACTION LIFECYCLE OF A BITCOIN")

Bitcoin transactions follow these steps:

Step 1: Creation

- User creates a transaction in their **wallet software**.
- Specifies inputs (UTXOs to spend) and outputs (recipient addresses, amounts).

Step 2: Signing

- Wallet software signs the transaction with the sender's **private key**.
- Signature proves ownership of the inputs being spent.

Step 3: Broadcast

- Signed transaction is broadcast to the **Bitcoin network**.
- Propagated using a **flooding algorithm** to reach all nodes.

Step 4: Transaction Pool (Memory Pool)

- Unconfirmed transactions are stored in the **transaction pool** (mempool) of nodes.
- Miners select transactions from this pool, prioritizing those with higher fees.

Step 5: Mining

- Miners validate transactions.
- Include them in a **candidate block**.
- Solve the **Proof-of-Work (PoW)** puzzle.
- The first miner to solve PoW broadcasts the new block with the included transactions.

Step 6: Block Confirmation

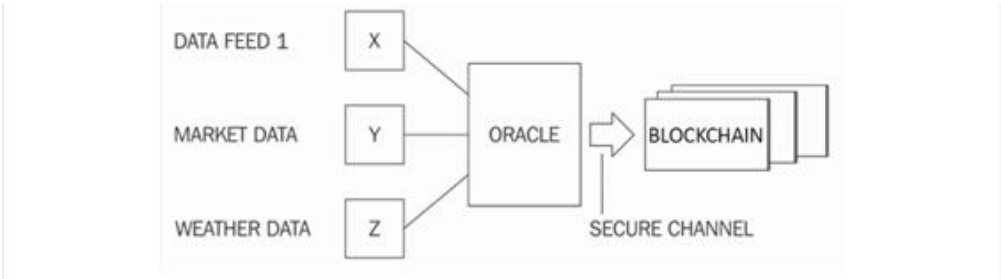
- The new block is verified and added to the blockchain.
- Other nodes propagate and validate the block.
- Transaction receives **confirmations** as new blocks build on top of it.

Step 7: Finality

- A transaction is considered secure after **multiple confirmations** (typically 6).
- Confirms that the transaction is permanently recorded on the blockchain and resistant to double-spending.

(create a block diagram considering all these 7 lifecycle parameters in box shaped vertical representation)

4. Explain a simplified model of an oracle interacting with smart contract on block chain.



- **Oracle** = an intermediary that fetches real-world data.
- **Blockchain** = where the data is sent, so that **smart contracts** can use it.

- **Secure channel** = ensures the integrity of data sent to blockchain.

Key point:

- *Smart contracts cannot access external data directly.*
- *Oracles bridge that gap securely.*

1) *Smart contracts are like computer program that runs on a blockchain.*
✅ They execute automatically on-chain.

2) *But they have one big limitation – they can't see or access the real-world information on their own.*
✅ They are **closed** to off-chain data by design.

3) *This is where oracles come in. Oracles act as messengers.*
✅ Oracles **fetch external data** and **deliver** it to the blockchain.

4) *They bring the data from the outside world and give it to smart contract.*
✅ This allows smart contracts to respond to real-world events.

5) *So in simple terms – Smart contracts can't go outside. Oracle: Someone who brings real-world data inside them.*
✅ Perfect metaphor for exam: *Oracles are like trusted messengers.*

6) *Example: If a smart contract needs a current gold price to work, the oracle will fetch that information from a trusted source and deliver it to the contract.*
✅ Real-world use-case example.

5. Explain the process of deploying smart contracts on a blockchain./ briefly discuss smart contracts on blockchain

Deploying smart contracts

Smart contracts may or may not be deployed on a blockchain, but it makes sense to do so on a blockchain due to the security and decentralized consensus mechanism provided by the blockchain. Ethereum is an example of a blockchain platform that natively supports the development and deployment of smart contracts. Smart contracts are self-executing contracts with the terms directly written into code. Deploying smart contracts on blockchains makes sense because the blockchain provides security and decentralized consensus.

Ethereum is the most notable blockchain that natively supports smart contracts, enabling the creation of decentralized applications (DApps).

In contrast, Bitcoin supports very basic forms of smart contracts through scripting features like transaction timelocks (nLocktime), and script operators like CHECKLOCKTIMEVERIFY (CLTV) and CHECKSEQUENCEVERIFY. These allow conditional transaction unlocking based on time or block height, for example, paying someone only after a certain time has passed. While limited, these are simple contract-like conditions.

Bitcoin scripts can also implement other basic contracts, such as requiring demonstration of a hash collision attack to unlock funds — a concept used in contests on the Bitcointalk forum.

Beyond Bitcoin and Ethereum, many other blockchain platforms support smart contracts, including Monax, Lisk, Counterparty, Stellar, Hyperledger Fabric, Neo, EOSIO, and Tezos. Smart contracts on these platforms can be written in domain-specific languages (DSLs) or general-purpose languages, but a key requirement is determinism — the contract must produce the same result everywhere it runs to ensure trust and reliability.

Common smart contract languages include:

• Solidity (Ethereum, running on the Ethereum Virtual Machine)

• JavaScript (Lisk)

• Go, Java, JavaScript (Hyperledger Fabric)

• C++ (EOSIO)

Security is critical for smart contracts. Numerous vulnerabilities have led to significant incidents, like the DAO attack on Ethereum. Consequently, smart contract validation and verification are active research areas aimed at improving reliability and security.

6. What is DAO?. Discuss

✅ What is DAO?

DAO stands for **Decentralized Autonomous Organization**.

- It is an organization that operates **without centralized control**, using **smart contracts on a blockchain** to **automate decision-making**, **governance**, and **fund management**.

- A **DAO** is a blockchain-based organization **governed by code** instead of traditional management. It uses **smart contracts** to automatically enforce rules and execute decisions, with no central authority.

Feature	Description
Decentralized	No central party or CEO; decisions made collectively by members.
Autonomous	Operates via smart contracts without manual intervention.
Transparent	All rules and transactions are recorded on the blockchain and are public.
Token-based Voting	Members hold tokens and vote on proposals; more tokens = more influence.
Immutable Code	Rules encoded in smart contracts can't be changed without consensus.

✅ How DAO Works:

1. **Smart Contracts Define Rules**
    - A DAO is launched with smart contracts that define how it works — who can vote, how funds are used, etc.
2. **Funding via Token Sale**
    - People fund the DAO by buying tokens. These tokens give them **voting rights**.

3. **Proposal System**
    - Anyone with tokens can create proposals (e.g., how to spend funds, what to build).
4. **Voting Mechanism**
    - Token holders vote on proposals. If a proposal passes, it is **automatically executed** by the smart contract.
5. **Execution**
    - The DAO automatically performs actions like transferring funds or updating code, based on the outcome of the vote.

✅ **Example: The DAO (2016)**

- One of the first DAOs launched on the Ethereum blockchain in 2016.
- It raised **$150 million** in Ether.
- Later **hacked due to a smart contract bug**, leading to a loss of $60 million.
- This incident led to the **Ethereum hard fork**, creating two chains:
    - **Ethereum (ETH)** and
    - **Ethereum Classic (ETC)**.

✅ **Advantages of DAOs:**

- ✅ No centralized corruption or manipulation.
- ✅ Transparent decision-making.
- ✅ Global participation and funding.
- ✅ Trust less operations (rules enforced by code).

✅ **Challenges / Limitations:**

- ⚠️ Bugs in smart contracts can be exploited.
- ⚠️ Difficult to modify once deployed.
- ⚠️ Legal recognition of DAOs is still unclear in many countries.
- ⚠️ Governance issues (e.g., token-based voting favours wealthy users).

7. Explain the different types of transactions in bitcoin system.

A Bitcoin transaction is the transfer of bitcoins from one user to another on the Bitcoin network. It is the fundamental action that moves value around the blockchain. The following are the types of transactions under SegWit or Segregated Witness a soft fork-based upgrade of the Bitcoin protocol that addresses weaknesses such as throughput and security in the Bitcoin protocol

1. Pay to Witness Public Key Hash (P2WPKH): This type of script is similar to the usual P2PKH, but the crucial difference is that the transaction signature used as a proof of ownership in ScriptSig is moved to a separate structure known as the "witness" of the input. The signature is the same as P2PKH but is no longer part of ScriptSig; it is simply empty. The PubKey is also moved to the witness field. This script is identified by a 20- byte hash. The ScriptPubKey is modified to a simpler format, as shown here:

• P2PKH ScriptPubKey:

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

• P2WPKH ScriptPubKey:

OP_0 <pubKeyHash>

2. Pay to Script Hash - Pay to Witness PubKey Hash (P2SH-P2WPKH): This is a mechanism introduced to make SegWit transactions backward-compatible. This is made possible by nesting the P2WPKH inside the usual P2SH.

3. Pay to Witness Script Hash (P2WSH): These scripts is similar to legacy P2SH but the signature and redeem script are moved to the separate witness field. This means that ScriptSig is simply empty. This script is identified by a 32-byte SHA-256 hash. P2WSH is a simpler script compared to P2SH and has just two fields. The ScriptPubKey is modified as follows:

• P2SH ScriptPubKey:

OP_HASH160 <pubKeyHash> OP_EQUAL

• P2WSH ScriptPubKey:

OP_0 <pubKeyHash>

4. Pay to Script Hash - Pay to Witness Script Hash (P2SH-P2WSH): Similar to P2SHP2WPKH, this is a mechanism that allows backward-compatibility with legacy Bitcoin nodes.

8. Write short note on i) Wallets ii) Smart Oracles.

Cryptocurrency Wallets Cryptocurrency wallets are essential tools that allow users to securely store, send, and receive digital assets like cryptocurrencies. They play a vital role in the security and usability of decentralized systems in the following ways:

• Private Key Management: Wallets securely store the user’s private keys, which are required to sign transactions. The private key is the critical secret that proves ownership and authorizes the transfer of funds. By securely managing these keys—either locally or with encryption— wallets prevent unauthorized access to assets.

• Transaction Signing: Wallets use private keys to digitally sign transactions, ensuring that only the rightful owner can initiate a transfer. This cryptographic signature guarantees authenticity and non-repudiation.

• User Interface and Access: Wallets provide a user-friendly interface to interact with the blockchain without requiring deep technical knowledge, which improves accessibility and adoption of decentralized applications.

• Security Features: Many wallets include additional layers of security such as multi-signature support, hardware wallet integration, and biometric authentication, all of which enhance the overall protection against theft and hacking.

Smart Oracles Smart oracles act as trusted intermediaries that feed external data into decentralized systems, particularly smart contracts, which operate on blockchain networks. Since blockchains cannot access off-chain information on their own, oracles are crucial for expanding their real-world functionality:

• Reliable Data Feeds: Oracles provide verified, tamper-proof data from external sources (e.g., price feeds, weather data, event outcomes) to smart contracts, enabling them to execute based on real-world conditions.

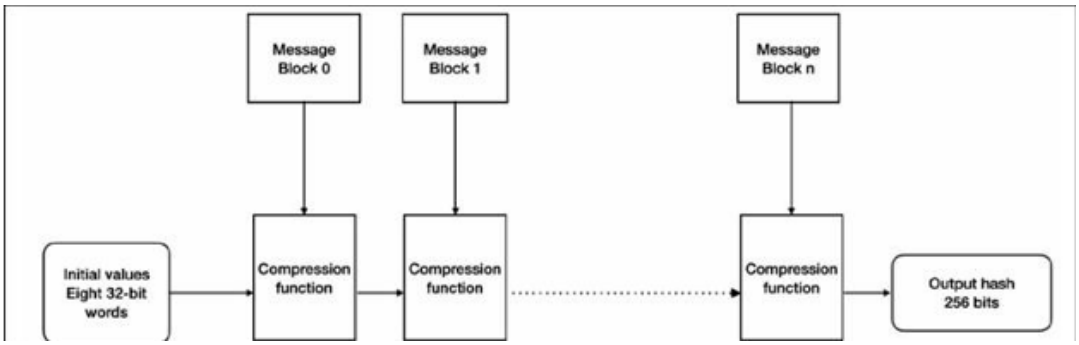
• Security and Trustworthiness: Smart oracles implement cryptographic proofs, decentralized data aggregation, or reputation systems to ensure the accuracy and integrity of data, mitigating risks such as manipulation or false information that could lead to incorrect contract execution.

• Automation and Functionality: By supplying dynamic, real-time information, oracles enable smart contracts to trigger complex automated processes — such as payments, insurance claims, or supply chain tracking — only when predefined external events occur.

• Decentralization of Data Sources: Some oracle designs use multiple independent data providers and consensus mechanisms to prevent reliance on a single point of failure, enhancing the security and robustness of decentralized applications

MODULE 2

1. Explain the design of SHA-256 algorithm



• SHA-256 has an input message size limit of 264 - 1 bits. The block size is 512 bits, and it has a word size of 32 bits. The output is a 256-bit digest. The compression function processes a 512-bit message block and a 256-bit intermediate hash value.

• There are two main components of this function: the compression function and a message schedule.

• The algorithm works as follows, in nine steps: Pre-processing

1. Padding of the message is used to adjust the length of a block to 512 bits if it is smaller than the required block size of 512 bits.

2. Parsing the message into message blocks, which ensures that the message and its padding is divided into equal blocks of 512 bits.

3. Setting up the initial hash value, which consists of the eight 32-bit words obtained by taking the first 32 bits of the fractional parts of the square roots of the first eight prime numbers. These initial values are fixed and chosen to initialize the process. They provide a level of confidence that no backdoor exists in the algorithm. Hash computation

4. Each message block is then processed in a sequence, and it requires 64 rounds to compute the full hash output. Each round uses slightly different constants to ensure that no two rounds are the same.

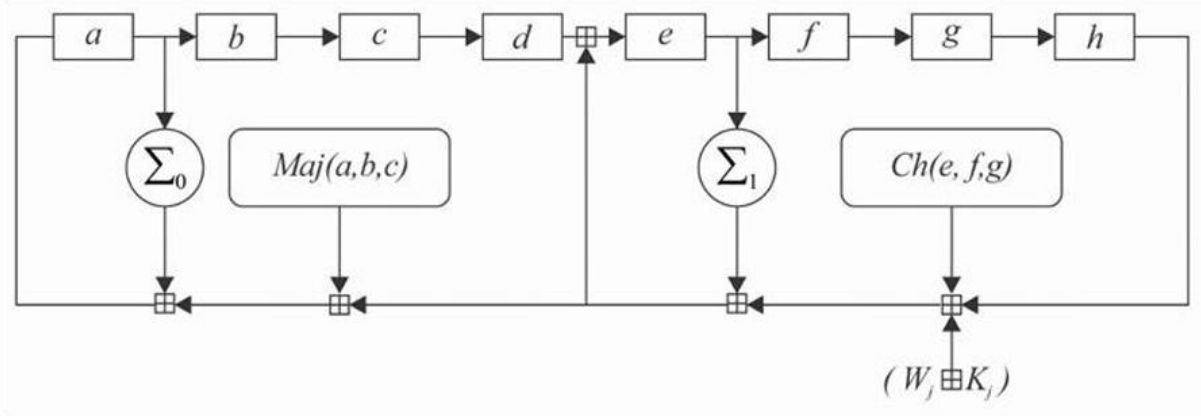
5. The message schedule is prepared.

6. Eight working variables are initialized.

7. The compression function runs 64 times.

8. The intermediate hash value is calculated.

9. Finally, after repeating steps 5 through 8 until all blocks (chunks of data) in the input message are processed, the output hash is produced by concatenating intermediate hash values



In the preceding diagram, a, b, c, d, e, f, g, and h are the registers for 8 working variables. Maj and Ch functions are applied bitwise. Σ0 and Σ1 perform bitwise rotation. The round constants are Wj and Kj , which are added in the main loop (compressor function) of the hash function, which runs 64 times.

2. What are the methods of decentralization? Discuss.

1. Disintermediation

- Definition: Removing intermediaries or trusted third parties from the system.
- Explanation:
    - Traditionally, centralized systems use intermediaries for validation, storage, and transmission of data (e.g., banks for money transfers).
    - In blockchain-based systems, data (like money transfers) can be directly exchanged peer-to-peer without intermediaries.

• This can be explained with the help of an example. Imagine you want to send money to your friend in another country.

• You go to a bank that will transfer your money to the bank in the country of your choice for a fee. In this case, the bank keeps a central database that is updated, confirming that you have sent the money.

• With blockchain technology, it is possible to send this money directly to your friend without the need for a bank. All you need is the address of your friend on the blockchain. This way, the intermediary is no longer required and decentralization is achieved by disintermediation.

• In the health industry, where patients, instead of relying on a trusted third party (such as the hospital record system) can be in full control of their own identity and their data that they can share directly with only those entities that they trust.

• As a general solution, blockchain can serve as a decentralized health record management system where health records can be exchanged securely and directly between different entities (hospitals, pharmaceutical companies, patients) globally without any central authority.

2. Contest-driven Decentralization

- Definition: Decentralization through competitive selection among multiple service providers.
- Explanation:
    - Instead of a single central entity, multiple providers compete to provide services based on:
        - Reputation
        - Previous performance
        - User reviews
        - Quality of service

• In the context of blockchain technology, a system can be envisioned in which smart contracts can choose an external data provider from a large number of providers based on their reputation, previous score, reviews, and quality of service. This method will not result in full decentralization, but it allows smart contracts to make a free choice based on the criteria just mentioned. This way, an environment of competition is cultivated among service providers where they compete with each other to become the data provider of choice.

Outcome:

- Does not achieve full decentralization, but ensures no single provider monopolizes the system.
- Encourages fair competition and increases system resilience

3. Give the different Platforms for decentralization.

Ethereum

- First blockchain to introduce a Turing-complete language for writing smart contracts.

- Uses Solidity as the primary programming language.

- Allows developers to build decentralized applications (DApps) and deploy smart contracts.

- Currency: Ether (ETH).

- Key Point: Public blockchain designed to be programmable, enabling endless use-cases

MaidSafe

- This is a project for the decentralized Internet introduced in 2006. This is not a blockchain, but a decentralized and autonomous network.

- MaidSafe provides a SAFE (Secure Access for Everyone) network that is made up of unused computing resources, such as storage, processing power, and the data connections of its users.

- The files on the network are divided into small chunks of data, which are encrypted and distributed randomly throughout the network. This data can only be retrieved by its respective owner.

- One key innovation of MaidSafe is that duplicate files are automatically rejected on the network, which helps reduce the need for additional computing resources needed to manage the load. It uses Safecoin as a token to incentivize its contributors.

Lisk

- Lisk is a blockchain application development and cryptocurrency platform. It allows developers to use JavaScript to build decentralized applications and host them in their respective sidechains.

- Lisk uses the Delegated Proof of Stake (DPOS) mechanism for consensus, whereby 101 nodes can be elected to secure the network and propose blocks. It uses the Node.js and JavaScript backend, while the frontend allows the use of standard technologies, such as CSS3, HTML5, and JavaScript.

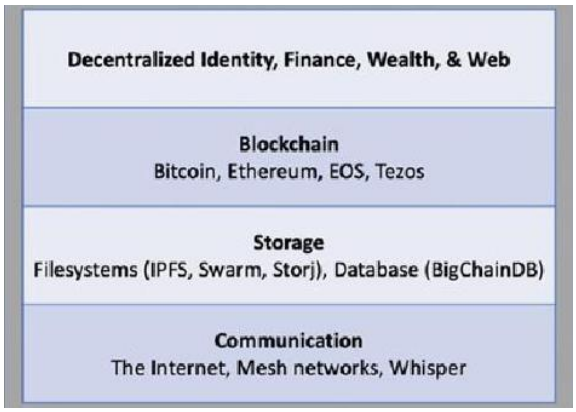
- Lisk uses LSK coin as a currency on the blockchain. Another derivative of Lisk is Rise, which is a Lisk-based DApp and digital currency platform. It offers greater focus on the security of the system.

EOS

- This is a blockchain protocol launched in January 2018, with its own cryptocurrency called EOS. EOS raised an incredible 4 billion USD in 2018 though its Initial Coin Offering (ICO).

- They key purpose behind EOS is, as stated by its founders, to build a decentralized operating system. Its throughput is significantly higher (approx. 3,996 transactions per second (TPS)) than other common blockchain platforms, such as Bitcoin (approx. 7 TPS) and Ethereum (approx. 15 TPS).

4.Explain full ecosystem decentralization of blockchain with a neat diagram.



1 Communication Layer

- The layer that handles **data transmission** between nodes.
- Traditional Internet relies on **centralized ISPs** and servers.
- This centralization means communication can be **censored, monitored, or shut down**.
- **Goal of decentralization**: Enable **peer-to-peer communication** without central authority.
- **Solutions:**
    - **Mesh Networks**:
        - Nodes connect directly without a central hub.
        - Example: **FireChat** (iOS app enabling offline messaging).
        - Useful where internet is censored or restricted.
    - **Whisper Protocol**:
        - Part of Ethereum stack for decentralized messaging.
        - Ensures **privacy** and **censorship resistance**.
- **Advantages**:
    - Resilient to shutdown.

    - Supports privacy and autonomy.
    - Decentralized control over data routes.

---

2 Storage Layer

- Blockchain **cannot store large data** (only transaction metadata).
- Need for decentralized, **fault-tolerant** storage.
- **Goals**:
    - Remove reliance on centralized data centers.
    - Ensure data availability even if some nodes fail.
- **Solutions**:
    - **IPFS (InterPlanetary File System)**:
        - Uses Distributed Hash Tables (DHT).
        - Stores content-addressed data with **Merkle DAGs**.
        - Efficient sharing and retrieval of files.
    - **Filecoin**:
        - Incentivized layer on top of IPFS.
        - Rewards nodes for storing data securely.
    - **Swarm**:
        - Ethereum-native decentralized storage system.
    - **BigChainDB**:
        - Decentralized database with blockchain features.
        - Supports high throughput.
    - **MaidSafe**:
        - SAFE Network (Secure Access For Everyone).
        - Uses **users' spare resources** for encrypted, distributed storage.
- **Advantages**:
    - Removes single points of failure.
    - Enables **global data access** without central control.
    - Supports **privacy** with encryption.

---

3 Computation Layer

- Blockchain enables **decentralized execution of business logic**.
- Traditionally, computation is **server-based** and centralized.
- **Goals**:
    - Decentralize processing.
    - Avoid reliance on single servers.
- **Solutions**:
    - **Ethereum**:
        - Smart contracts run on all nodes.
        - Uses consensus to validate execution.
    - **Golem**:
        - Decentralized marketplace for computing power.
        - Users share/rent CPU resources.
    - **EOS, Lisk**:
        - Support scalable smart contract execution.
        - Sidechains and high throughput.
- **Advantages**:
    - Removes trust in a single provider.
    - Increases resilience and fairness.
    - Enables global decentralized apps (DApps).

---

4 Identity and Wealth Layers

- Identity management is typically centralized (e.g., government IDs).
- Wealth is managed by centralized banks.
- **Goals**:
    - Give users **self-sovereign identity**.
    - Enable **decentralized finance (DeFi)**.
- **Solutions**:.
    - **DAOs (Decentralized Autonomous Organizations)**:
        - Organizations governed by code.
        - Voting and decision-making on-chain.

- **DeFi Platforms**:
    - Lending, borrowing, trading without intermediaries.
    - Examples: Uniswap, Aave.

- **Advantages**:
    - Users control their own identities and funds.
    - Trustless interactions without central authorities.
    - Enables **global, inclusive financial systems**.

5. Explain Asymmetric cryptography with suitable diagrams.

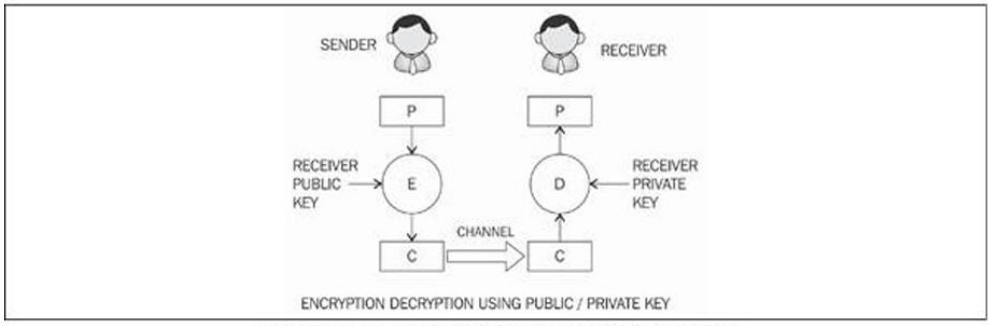


Figure 4.1: Encryption/decryption using public/private keys

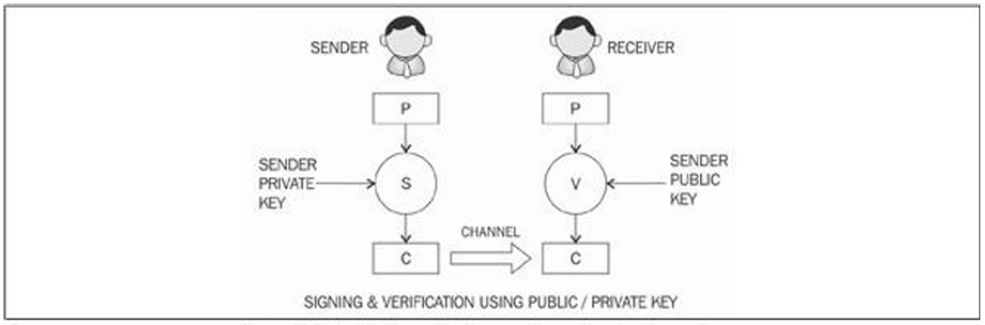


Figure 4.2: Model of a public-key cryptography signature scheme

✅ Asymmetric Cryptography

🔹 Definition

Asymmetric cryptography (or **public-key cryptography**) is a cryptographic system that uses **two mathematically related keys**:

- **Public Key**:
    - Shared openly with anyone.
    - Used for encrypting data or verifying signatures.

- **Private Key**:
    - Kept secret by the owner.
    - Used for decrypting data or creating digital signatures.

Unlike **symmetric cryptography** (where the same key is used for both encryption and decryption), asymmetric cryptography **separates** these operations to improve security, especially over untrusted networks.

✅ Key Characteristics

- **Two distinct keys**:
    - Keys are mathematically linked.
    - Impossible to derive private key from the public key (for practical purposes).
- **One-way operations**:
    - Encrypt with one key, decrypt with the other.
- Enables **secure communication** even between parties that have never met before.

✅ Working Principle

🔹 1. Encryption and Decryption

- **Encryption**:
    - Sender uses **receiver's public key** to encrypt the message.
- **Decryption**:
    - Only the receiver can decrypt using their **private key**.

Result: Confidential communication even over insecure channels.

Example:

Alice wants to send Bob a secret message.

- Bob gives Alice his public key.
- Alice encrypts the message with Bob’s public key.
- Only Bob can decrypt using his private key.

🔹 2. Digital Signatures

- Sender **signs** a message with their **private key**.

- Anyone can **verify** the signature using the sender's **public key**.
- Ensures:
    - **Authentication**: Confirms the sender's identity.
    - **Integrity**: Confirms message wasn't altered.
    - **Non-repudiation**: Sender can't deny sending.

Example:

Bob signs a transaction with his private key.
The blockchain network verifies the signature with Bob's public key.

✅ **Mathematical Principle**

- Based on **one-way mathematical functions** (e.g., factoring large primes, elliptic curves).
- Easy to perform encryption/decryption with the key pair.
- Hard (computationally infeasible) to reverse-engineer private key from public key.

✅ **Applications in Blockchain**

- **Wallet addresses** are derived from public keys.
- **Transaction signing** ensures:
    - Ownership of funds.
    - Prevention of tampering.
- Nodes **verify** transactions using the sender's public key.
- Removes need for a **trusted central authority**.

✅ **Advantages**

✅ Secure communication without sharing secrets.
✅ Enables **authentication**.
✅ Supports **non-repudiation** (cannot deny having sent a message).
✅ Key distribution is simpler than symmetric systems.
✅ Vital for **digital certificates**, **SSL/TLS**, and **blockchain**.

6. Explain Elliptic Curve Digital signature algorithm (ECDSA) with a practical example.

🔹 **Definition**

Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm (DSA) which uses **Elliptic Curve Cryptography (ECC)**.

It is an **asymmetric cryptographic algorithm** used to generate and verify digital signatures, providing:

- **Authentication** (confirm sender's identity).
- **Integrity** (ensure message not altered).
- **Non-repudiation** (sender cannot deny signing).

✅ Key Features of ECDSA

- Based on **elliptic curve mathematics** over finite fields.
- Provides **high security** with **smaller key sizes** compared to RSA.
    - E.g., 256-bit ECDSA ≈ 3072-bit RSA.
- Efficient computation—important for blockchain, IoT, and mobile.
- Widely used in **Bitcoin, Ethereum** and other cryptocurrencies.

✅ How ECDSA Works – Main Steps

1 Key Generation

- Choose an elliptic curve **E** over a finite field.
- Select a **base point G** on the curve.
- Private key (d): a randomly selected integer.
- Public key (Q): calculated as **Q = d * G**.

---

2. Signing a Message

Given a message **M**:

1. Hash the message **H = Hash(M)**.
2. Choose random integer **k** in [1, n−1].
3. Calculate point **$(x_1, y_1) = k * G$**.
4. Compute **$r = x_1 \bmod n$**.
5. Compute **$s = k^{-1} (H + d \cdot r) \bmod n$**.
6. Signature = **(r, s)**.

---

3. Verifying the Signature

Given signature **(r, s)** and public key **Q**:

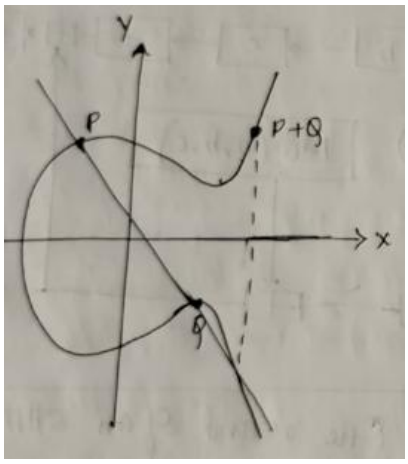
1. Hash the message **H = Hash(M)**.
2. Compute **$w = s^{-1} \bmod n$**.
3. Compute **$u_1 = H \cdot w \bmod n$**.
4. Compute **$u_2 = r \cdot w \bmod n$**.
5. Compute **$(x_1', y_1') = u_1 \cdot G + u_2 \cdot Q$**.
6. Verify if **$r \equiv x_1' \bmod n$**.

✅ If it holds, signature is valid!

✅ Practical Example (Blockchain / Bitcoin)

- In **Bitcoin**, ECDSA is used to **sign transactions**.
- A user's **private key** generates the signature proving ownership of coins.
- Example:
  - Alice wants to send 1 BTC to Bob.
  - Alice's wallet **signs the transaction** with her ECDSA private key.
  - The Bitcoin network **verifies** the signature using Alice's **public key** (in her address).
  - Ensures:
    - Only Alice can authorize spending.
    - No tampering is possible.

Diagram for point addition in ECC(write the same answer as above)

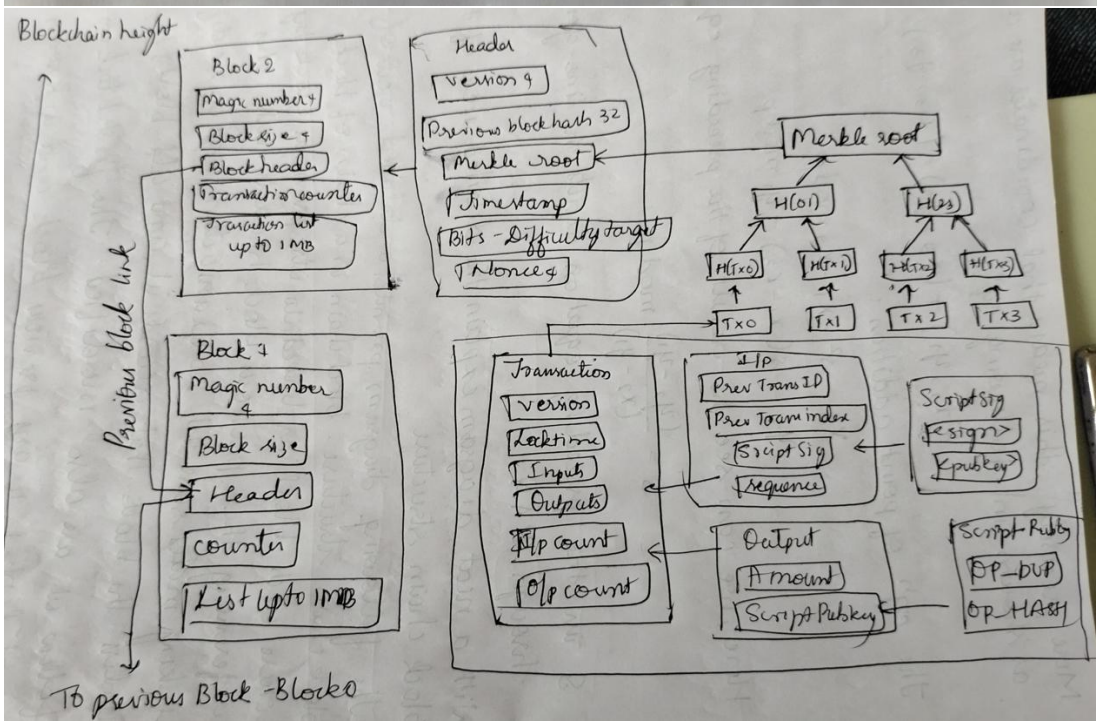


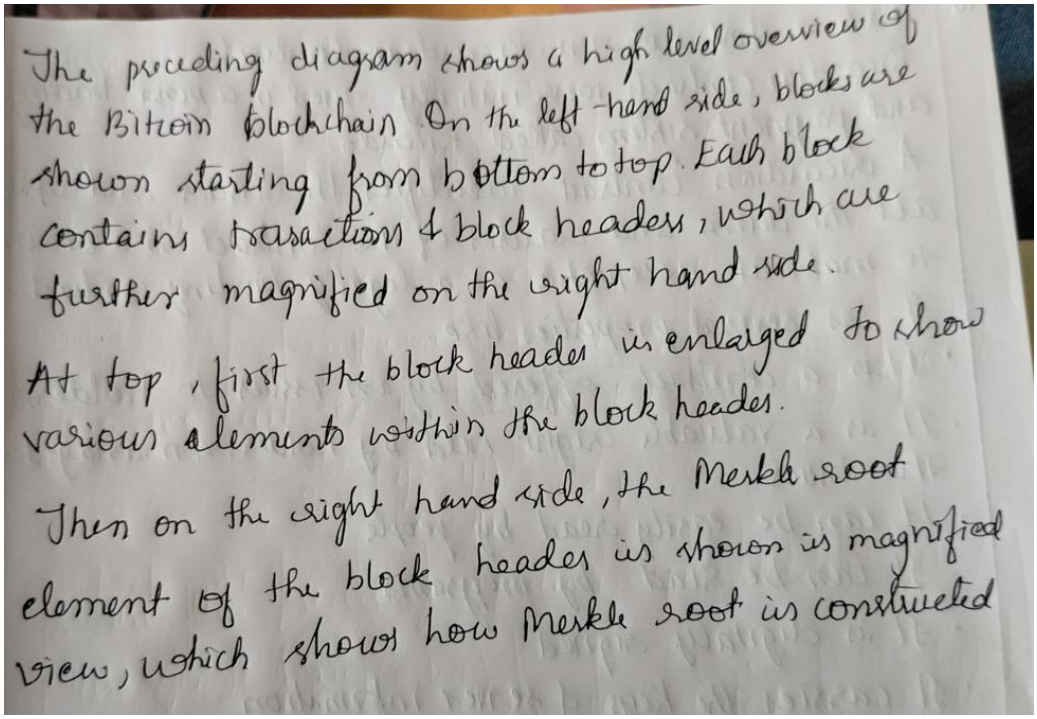
MODULE 1

1.Define Blockchain. Explain the network view/detailed view of Blockchain with a neat diagram.

9 With a neat diagram explain the detailed view of the block chain structure.

The following diagram provides a detailed view of the blockchain structure. Blockchain is a chain of blocks where each block is linked to its previous block by referencing the previous block header's hash. This linking makes sure that no transaction can be modified unless the block that records it and all blocks that follow it are also modified. The first block is not linked to any previous block and is known as the genesis block.





Blockchain is a *peer-to-peer distributed ledger* that is **cryptographically secure**, **append-only**, **immutable**, and **updateable only via consensus among peers**.

It is:

- A decentralized system with **no single point of control**.
- A **distributed database** that stores records in linked blocks.
- A ledger maintained collectively by **multiple nodes** over a network.
- Resistant to **tampering** due to cryptographic hashing.

✅ **Alternative Business Definition (also in notes)**

Blockchain is a **platform** where peers can **exchange value using transactions** without the need for a **central trusted arbitrator**.

✅ **Key Features (VTU Points)**

✅ Cryptographically secure – relies on hashes and digital signatures.
✅ Append-only – new data can be added but not modified.
✅ Immutable – once data is recorded, it is extremely hard to change.
✅ Distributed consensus – ensures all nodes agree on the same state.
✅ Peer-to-peer – no central server; all nodes participate equally.

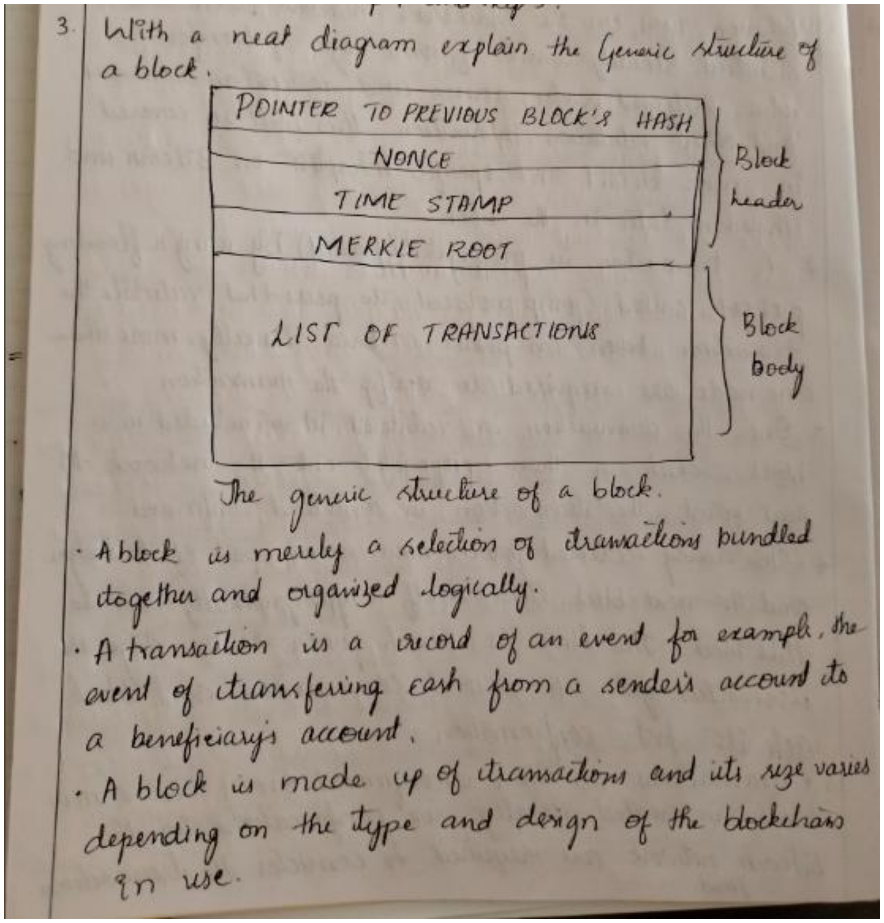
✅ Network View of Blockchain

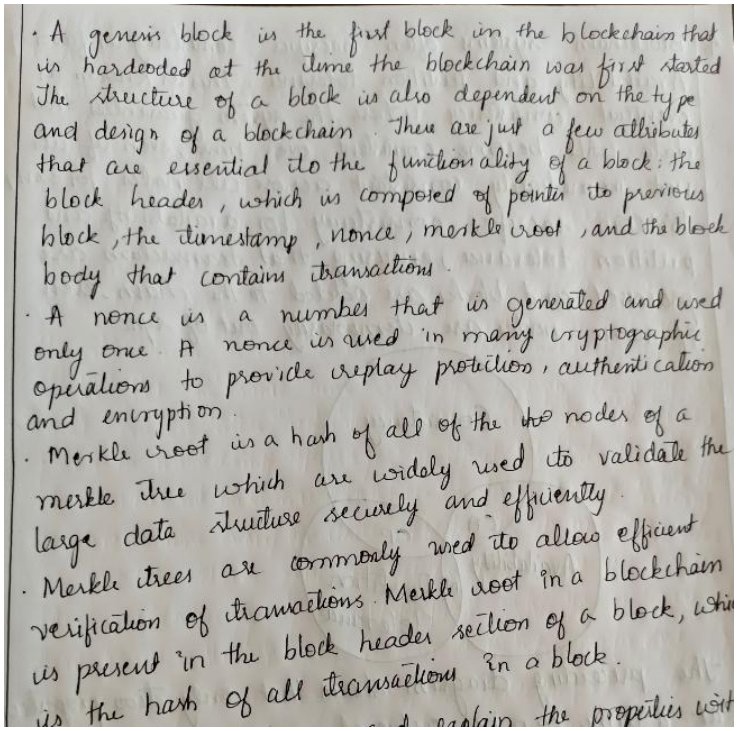
In the **network view**, blockchain is seen as a **layer on top of the Internet**, similar to how protocols like **HTTP, SMTP, or FTP** run over **TCP/IP**.

Key aspects:

- Blockchain network is a **peer-to-peer (P2P) network**.
- Each **node**:
  - Holds a **copy of the ledger**.
  - Participates in **transaction validation**.
  - Takes part in **consensus mechanisms**.
- Nodes communicate directly **without intermediaries**.
- The network uses **cryptographic techniques** to ensure **integrity and trust**.
- Transactions are **broadcast** and **validated** by multiple nodes.
- Consensus algorithms (like PoW, PoS) ensure **agreement** on transaction order and validity.

2.List and explain the generic elements of Blockchain.





3. Define Consensus. List the requirements and types of Consensus mechanism.

Consensus is the process of agreement among distrusting nodes on a final state of data in a distributed system.

It ensures that all honest participants in the network agree on the same version of the ledger without the need for a central authority.

✅ In blockchain, consensus:

- Resolves conflicts about transaction order.
- Ensures a single shared truth.
- Enables trust less systems.

✅ Requirements of a Consensus Mechanism

(From the notes, in exam-friendly bullet points)

1.Agreement

- All honest nodes must agree on the same value.

2. Termination

- All honest nodes eventually reach a decision and stop.

3. Validity

- The agreed-upon value must have been proposed by at least one honest node.

4. Fault Tolerance

- Must work even with faulty or malicious (Byzantine) nodes.

5. Integrity

- No node makes the decision more than once in a single consensus cycle.

✅ Types of Consensus Mechanism

(from the notes)

✅ 1. Proof-based / Leader-based / Nakamoto consensus

- A leader is elected to propose the final value.
- Example: Proof of Work (PoW) in Bitcoin.
- Nodes solve computational puzzles to propose blocks.

✅ 2. Byzantine Fault Tolerance (BFT)-based

- Traditional approach with rounds of voting.
- Nodes exchange signed messages.
- Agreement reached when a certain number of messages are received.
- Example: Practical Byzantine Fault Tolerance (PBFT).

✅ Other types (as per notes):

- Proof of Stake (PoS)
    - Selection based on stake in the network.
- Proof of Elapsed Time (PoET)
    - Uses trusted hardware to enforce wait times.
- Proof of Importance
    - Considers stake and transaction activity.
- Proof of Activity

- Proof of Activity is a hybrid blockchain consensus mechanism that combines elements of Proof of Work (PoW) and Proof of Stake (PoS).
- It is designed to increase security and reduce energy consumption compared to pure PoW.

4. List and explain the Consensus Algorithms or mechanisms.

1.Proof-based / Leader-based (Nakamoto Consensus)

- Used in Bitcoin.
- Miners compete to solve cryptographic puzzles (Proof of Work).
- The first to solve proposes the next block.
- Secure against Sybil attacks.
- Drawback: High energy consumption.

✅ Examples: Bitcoin, Litecoin.

2.Byzantine Fault Tolerance (BFT)-based

- Traditional distributed systems approach.
- Nodes exchange signed messages in rounds.
- When enough signatures collected, consensus is reached.
- Tolerates Byzantine (malicious) nodes.
- Low energy use, but needs known participants.

✅ Examples: PBFT, Hyperledger Fabric.

---

3. Proof of Stake (PoS)

- No mining or energy-intensive puzzles.
- Block proposer chosen based on stake in the system.
- Higher stake = higher chance to propose next block.
- Reduces energy costs.

✅ Examples: Ethereum 2.0, Cardano.

---

4. Delegated Proof of Stake (DPoS)

- Stakeholders vote to elect delegates.
- Elected delegates validate transactions and produce blocks.
- Improves efficiency and speed.
- Can be less decentralized if few delegates.

✅ Examples: EOS, BitShares.

---

5. Proof of Elapsed Time (PoET)

- Designed for trusted hardware (Intel SGX).
- Nodes wait a random, verifiable time.
- Shortest wait time wins.
- Reduces energy usage.

✅ Example: Hyperledger Sawtooth.

---

6. Deposit-based Consensus

- Nodes provide a security deposit to participate.
- Misbehaving nodes can lose their deposit.
- Incentivizes honest behavior.
- Helps avoid Sybil attacks.

✅ Example: Variants used in PoS systems.

---

7. Proof of Importance

- Extends PoS by adding activity measures.
- Factors in stake and transaction behavior.
- Active, trustworthy users more likely to propose blocks.

✅ Example: NEM blockchain.

---

8. Federated Byzantine Consensus (Federated BFT)

- Each node maintains a list of trusted peers.
- Consensus only among quorums of trusted nodes.
- Improves speed while maintaining fault tolerance.

✅ Example: Stellar Consensus Protocol.

---

9. Reputation-based Mechanisms

- Leaders are chosen based on reputation earned over time.
- Reputation built by past honest behavior.
- Mitigates attacks by favoring trustworthy nodes.

✅ Examples: Used in permissioned networks.

5. Explain benefits, features and limitations of blockchain.

1. Features of Blockchain

(As in notes, Module 1, p. 20–23)

✓ Distributed Ledger

- Shared ledger accessible to all participants.
- Removes need for central authority.

✓ Immutability

- Records once added cannot be easily changed.
- Achieved via cryptographic hashes and linking of blocks.

✓ Decentralization

- No single point of control or failure.
- Each node maintains a copy of the ledger.

✓ Transparency

- All participants can view the ledger.
- Increases trust among participants.

✓ Consensus Mechanism

- Transactions validated via agreement among nodes.
- Examples: PoW, PoS, PBFT.

✓ Security

- Cryptographic techniques (hashing, digital signatures).
- Protects data integrity and authenticity.

✓ Smart Contracts

- Automated programs stored on blockchain.
- Self-execute when predefined conditions are met.

✓ Uniqueness

- Each transaction is unique.
- Prevents double spending.

✓ High Availability

- Peer-to-peer replication ensures system works even if some nodes fail.

✓ Smart Property

- Link digital/physical assets to blockchain for unique ownership.

---

2. Benefits of Blockchain

(From notes, p. 30–31)

✅ Decentralization

- Removes need for trusted third parties.
- Peer-to-peer transactions without intermediaries.

✅ Transparency and Trust

- All transactions are visible to all network participants.
- Reduces fraud and corruption.

✅ Immutability

- Extremely difficult to alter past records.
- Ensures data integrity.

✅ High Availability

- Even if some nodes fail, network stays operational.

✅ Security

- Strong cryptographic techniques secure data.
- Resistant to tampering and attacks.

✅ Cost Savings

- No intermediaries reduces transaction fees.
- Simplifies processes across industries.

✅ Faster Transactions

- Reduces settlement times in finance.
- Real-time verification and finality.

✅ Simplification of Business Processes

- Single shared ledger reduces data silos.
- Easier auditing and compliance.

✅ Smart Contracts

- Automates execution of agreements.
- Reduces need for manual intervention.

✅ Enables New Business Models

- Tokenization of assets.
- Decentralized Finance (DeFi).

---

3. Limitations / Challenges of Blockchain

(From notes, p. 31–32)

❗ Scalability

- Limited transaction throughput (block size limits).
- High latency for confirmation.

❗ Energy Consumption

- PoW mining is highly energy intensive.
- Environmental concerns.

❗ Storage Issues

- Each node stores entire blockchain.
- Size grows over time.

❗ Time-Consuming Transactions

- Mining requires solving puzzles → slows confirmation.
- Not ideal for high-speed systems.

❗ Immaturity

- Still an evolving technology.
- Needs broader adoption and standardization.

❗ Legal and Regulatory Issues

- Varying regulations in different countries.
- Some countries ban cryptocurrencies.

❗ Privacy Concerns

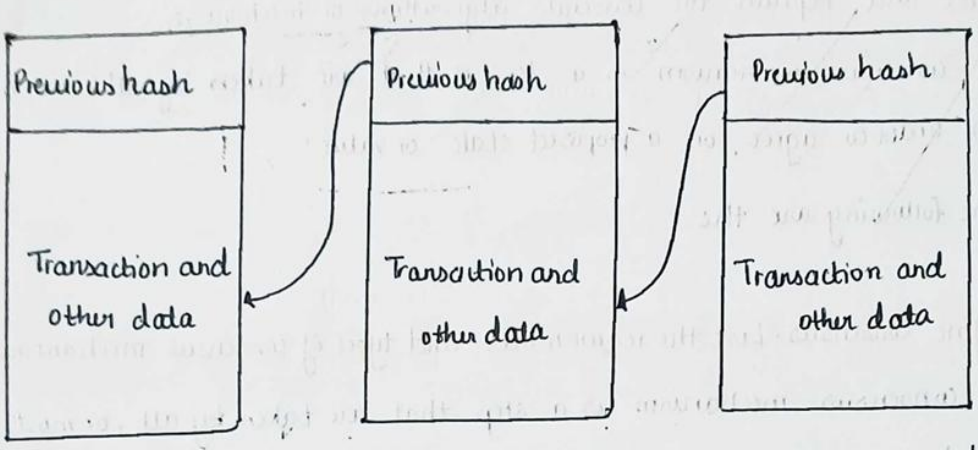
- Public blockchains lack confidentiality.
- Transparency can conflict with data privacy needs.

❗ Adaptability

- Integration with existing systems can be complex.
- Not all industries ready for decentralization.

6) List and explain the generic elements of Blockchain (6)

→



1) Addresses: Blockchain addresses are unique identifiers for senders and recipients in transaction often derived from public key. while addresses can be reused, its recommended to generate a new one for each transaction to enhance privacy.

2) Transaction: A transaction is a fundamental unit of Blockchain, which represents a tranfer of value from one address to another

3) Block: A block is composed of multiple transactions and some other elements such as the previous block hash, timestamp and nonce

4) Peer to peer network: This is a network topology whereby all peer can communicate with each other and send and receive message

5) Programming language: Turing complete programming language is a desireabl feature of blockchains

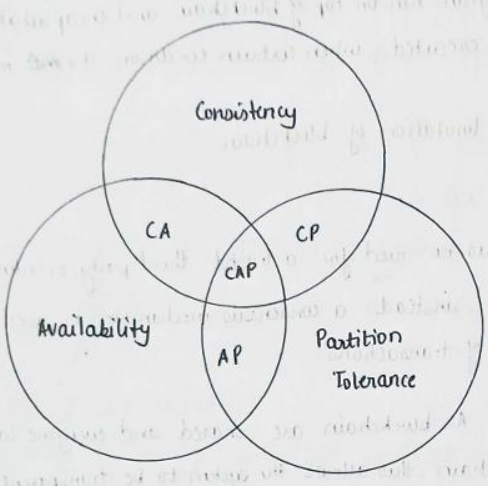
6) Virtual machine: A virtual machine allows Turing complete code to be run on a blockchain

7) State machine: A blockchain can be viewed as a state transition mechanism

8) Nodes: Nodes propose and validate transactions, pefom mining and ensures consensus using protocol like Proof of Work.

(8) With a neat diagram, explain CAP theorem.

→



1) The CAP theorem is also known as Brewers theorem, was introduced by Eric Brewer in 1998

2) This theorem states that any distributed system cannot have consistency, availability and partition tolerance simultaneously.

3) Consistency: is a property that ensures all the nodes in distributed system have single, current and identical copy of data

4) Availability: means that nodes in system are up, accessible for use and are accepting incoming request and responding with data without any failures

5) Partition Tolerance: ensures that if a group of nodes is unable to communicate with other nodes due to network failures, the distributed system continues to operate correctly. This occur due to network and node failure

---

6) If we opt for CP (consistency and partition tolerance), we sacrifice availability

7) If we opt for AP (availability and partition tolerance) we sacrifice consistency

8) If we opt for AC (availability and consistency), we sacrifice partition tolerance