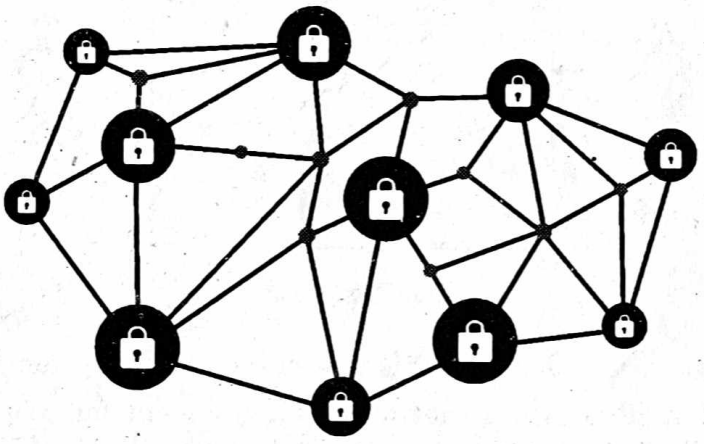
1.1 INTRODUCTION TO BLOCKCHAIN

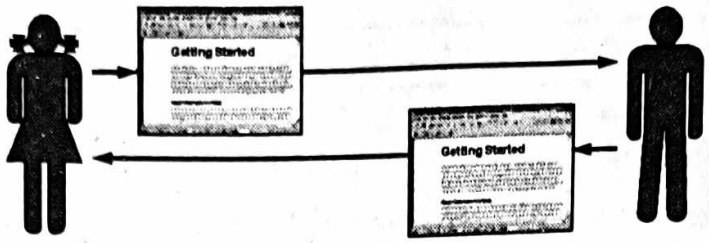
GQ. Define blockchain and explain it with a suitable example.

- By definition, blockchain is a *decentralized computation and information sharing platform* that enables *multiple authoritative domains*, who do not trust each other, to *cooperate*, *coordinate*, and *collaborate* in a rational decision making process.
- The keyword that we have in the abovementioned definition is decentralized, which is an important aspect of blockchain.
- Computation and information sharing platform is the next keyword that will help in specifying blockchain broadly to be a decentralized database, which helps in cooperation between multiple authoritative domains.
- This technology is useful when multiple parties or individuals want to cooperate with each other and they want to come to a common platform so that they can share information amongst themselves.



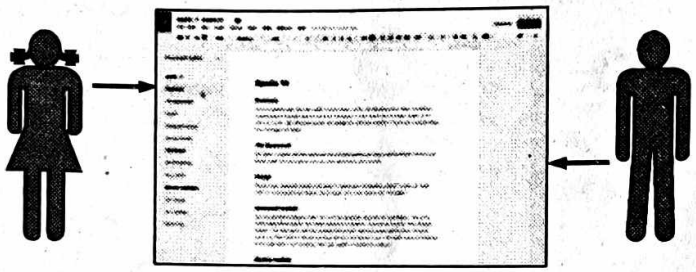
(1A1)**Fig. 1.1.1 : Structure of a decentralized network**

- Multiple authoritative domains do not trust each other, so what blockchain does is it combines these multiple authoritative domains in a common platform so that they can cooperate, coordinate, and collaborate in application development process or business intelligence process.
- Traditional way of sharing documents, which we do using Microsoft Word. For example, Person A wants to share some document with Person B. So, ideally what Person A will do is he/she will write the content in a Microsoft Word document and then he/she will share that document with Person B.
- Person B will first receive the document sent by Person A and then Person B will update the document with his/her content and share the document again with Person A.



(1A2)**Fig. 1.1.2**

- This was the traditional way of cooperation between Person A and Person B when they wanted to write something in a shared document. In this specified process, one of the main disadvantages is that Person A and Person B will not be able to simultaneously edit the document. The ideal solution where simultaneous editing of the document can be done is with the help of shared Google docs where both Person A and Person B can edit or write the document simultaneously.
- However, the problem of shared Google doc platform is that the environment is still centralized.



(1A3)**Fig. 1.1.3**

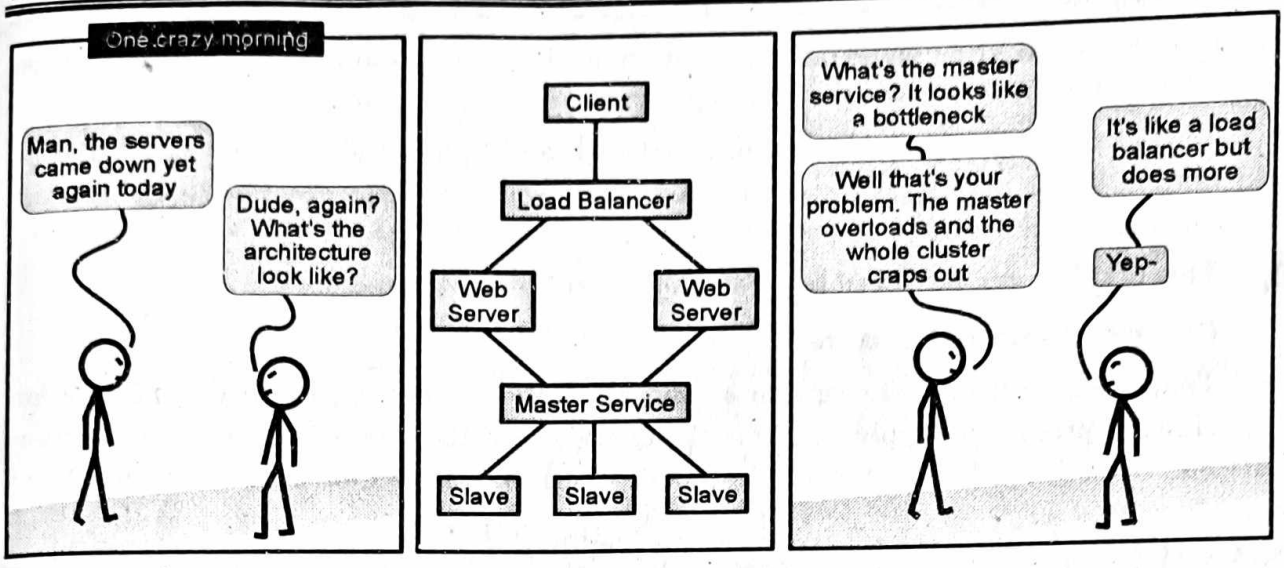
- Now, the question arises that does this type of centralized system harm or what is the disadvantage that if we use a centralized environment for cooperation during this kind of information sharing platform.

1.1.1 Problems with a Centralized System

GQ. What problems are associated with a centralized system?

GQ. Differentiate between centralized, decentralized, and distributed architecture.

- The major problem of a centralized system is that it works as a single point of failure. For example, if sufficient bandwidth is not available to load the Google doc, then a person will not be able to edit the document. You will have to wait until you connect to the Internet and load the Google doc platform. The other problem that could arise is what if Google's server or your computer crashes.



(1A4)**Fig. 1.1.4**

- In that case, the entire information gets lost or even if you want to take a backup, you will have to load the backup and only after loading the backup, you will be able to process it further. Because of the abovementioned reasons, it is necessary to shift from a centralized platform to a distributed platform.
- In a distributed platform, there are three different types of architecture :

(1) Centralized architecture
(2) Decentralized architecture
(3) Distributed architecture

(1) Centralized architecture

- There is a central coordination system and every node is connected to the central coordination system and whatever information the nodes want to share, the information shall be shared by the central coordination system.
- The problem with this architecture is that if the central coordination system fails, then all of the individual nodes will get disconnected.

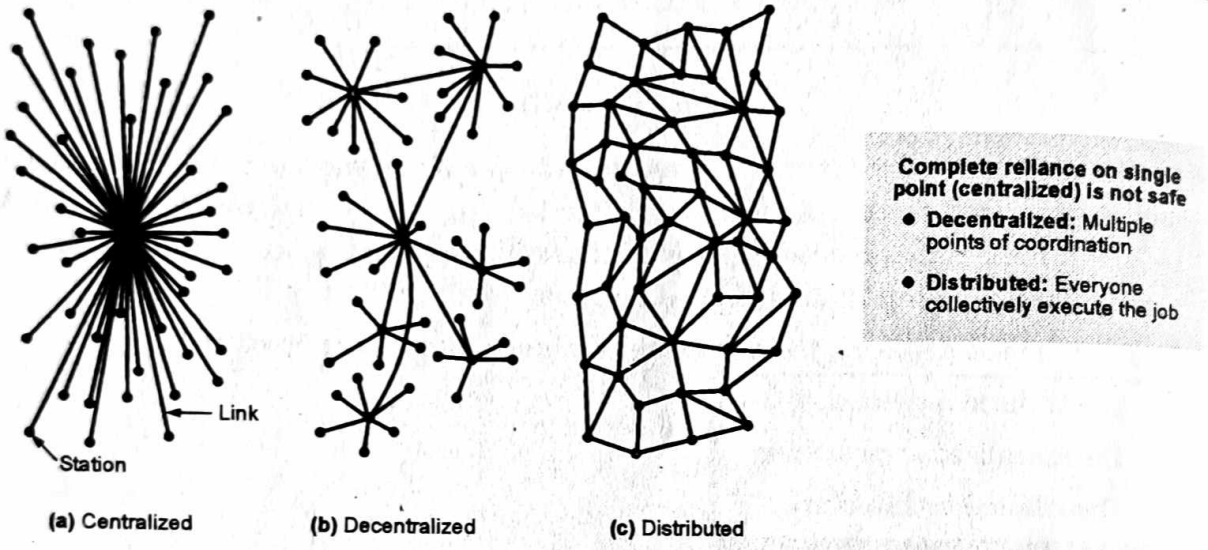
(2) Decentralized architecture

- Here, there are few coordinators rather than a single coordinator, and all these coordinators cooperate with each other, and the individual nodes, they are connected to these coordinators.
- In this architecture, if a particular coordinator fails or multiple coordinators fail, then the individual nodes whose coordinators have failed can get connected to other coordinators that are working and can share the information or perform the operation with the available coordinators.

- In this type of architecture, the advantage is that multiple number of failures can be tolerated until the network becomes disconnected.
- This architecture works on top of a network and because of multiple simultaneous failure if the network gets partitioned or the network gets disconnected, then the individual nodes will not be able to cooperate with each other.
- This problem can be solved by a complete distributed architecture.

(3) Distributed architecture

- There is no centralized coordinator and all the nodes participate in the information sharing process or application development where they coordinate with each other and collectively develop the application or share the information amongst themselves.



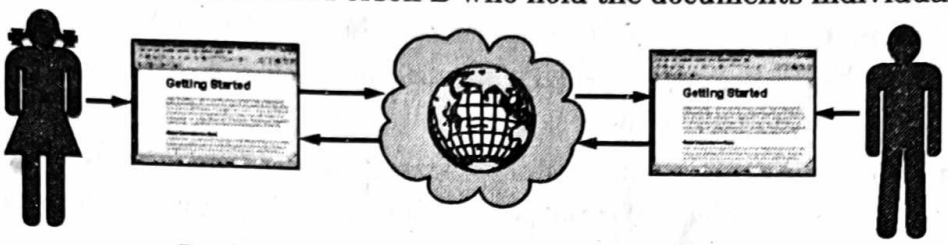
(1A5)**Fig.1.1.5**

- These are the advantages or disadvantages of a decentralized platform and distributed platform over a centralized platform.
- However, in the today's world, it has been observed that centralized architectures are good but are not scalable and not robust to failure.
- Therefore it is essential that we move from a centralized architecture to a decentralized architecture or distributed architecture.
- Blockchain is a platform that supports a decentralized platform or distributed platform where information can be shared amongst users in a trust worthy manner.

1.1.2 An Ideal Solution using Blockchain

- In a blockchain platform, Person A has his/her own copy of the document, and Person B too has his/her own copy of the document. Person A and Person B can simultaneously write to their own document and there is a network in between that

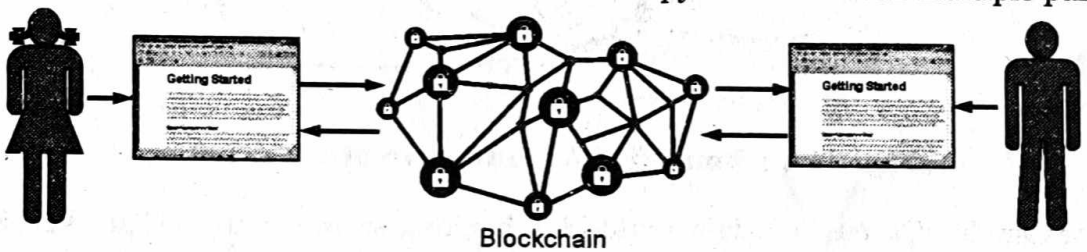
has the task to ensure that information consistency is maintained between the documents with Person A and Person B who hold the documents individually.



Everyone edits on their local copy of the document - the Internet takes care of ensuring consistency

(1A6)**Fig. 1.1.6**

- This is an ideal use case where blockchain platform can be used. This platform, which is spanned over a network, will help in creating this type of coordination where Person A will keep his/her own copy of the document and Person B will keep his/her own copy of the document.
- Both of them can independently write their personal copy and then the blockchain platform will ensure that they are entering inside the document are getting synchronized with each other, and with time, both of them will be able to see the most updated copy or they will be able to update over the most updated copy.
- This is the advantage of a blockchain technology over a complete centralized architecture or an architecture where a shared copy exists between multiple parties.



Blockchain

A decentralized database with strong consistency support

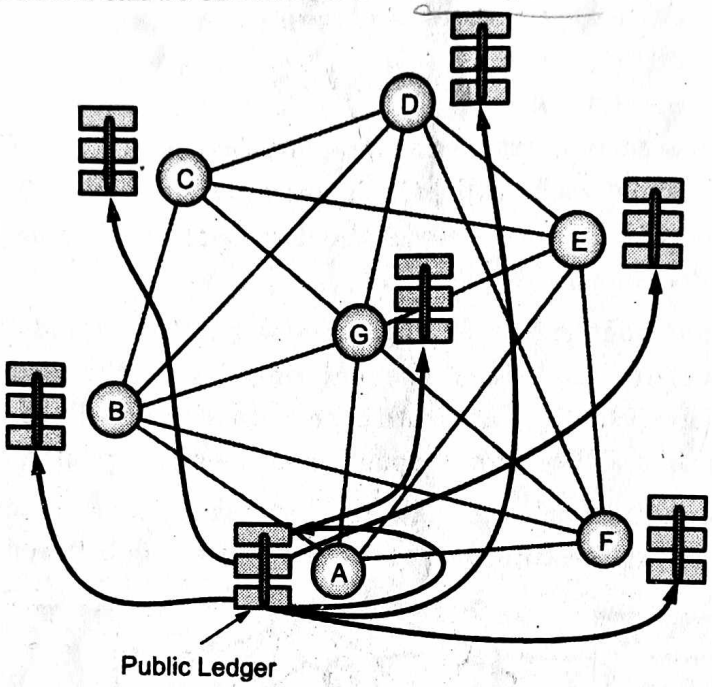
(1A7)**Fig. 1.1.7**

1.1.3 Simplified Architecture of a Blockchain

GQ. Explain with a suitable diagram, the simplified architecture of a blockchain.

- In a typical blockchain architecture, every individual node maintains a local copy of the blockchain (i.e., a local copy of the global data sheet). The systems task is to ensure that all these individual copies are consistent with each other. Consistency means that the local copies that every node has are identical and these copies are always updated based on the global information.

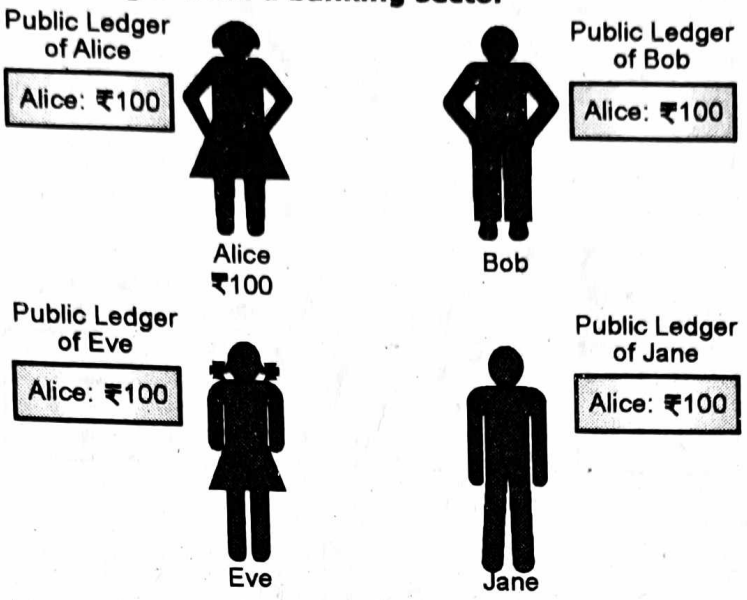
- For instance, if node A wants to enter some information to its blockchain, then the entered information will get updated to all the copy of the blockchain that every node (i.e., nodes B, C, D, E, F, and G) possess. This is the architectural platform of blockchain which supports strong consistency among the local replica/local information that every node has.
- The local information is called a public ledger. A public ledger works like a database where it contains historical information which is available to everyone and this type of historical information can be utilized for future computation.



(1A8)**Fig. 1.1.8 : Simplified Architecture of a Blockchain**

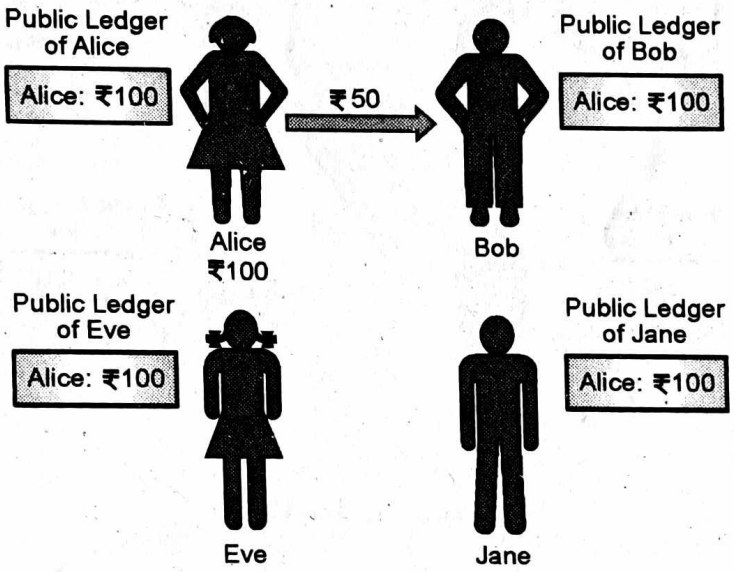
- An example of a public ledger could be a banking transaction. You are keeping the banking transactions inside a public ledger and the old transactions which are there are basically used to validate the new transactions.
- In a typical banking system, we usually maintain a passbook and the bank works like a centralized authority which stores all our transaction information and whenever an individual visits a bank and making a transaction, the bank validates everything with the centralized information that it has.
- Now, in case of a pubic ledger, there is a shift from a centralized banking system to a decentralized banking system where every individual has his/her own copy of the global transactions, which not only synchronized but also consistent, and whenever an individual attempts to make a new transaction, during that time the new transaction is validated against the old transaction that is already there inside a public ledger.

☞ **Example of a public ledger from a banking sector**



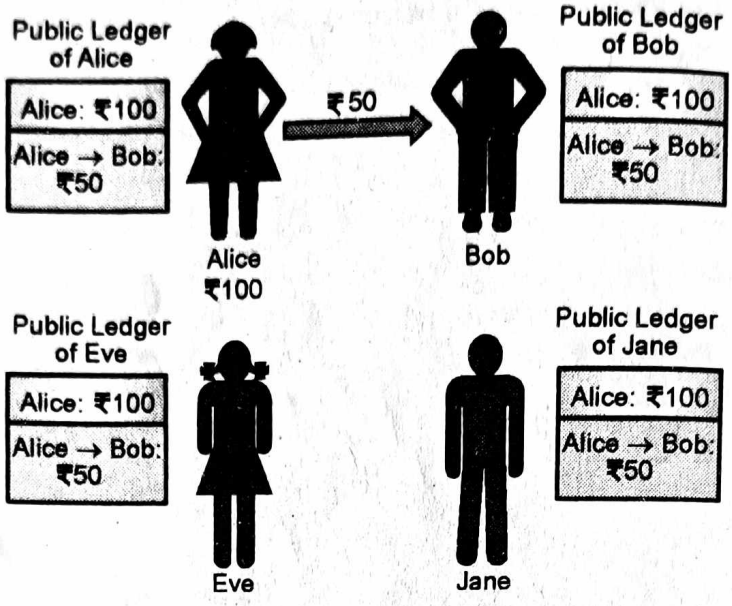
(1A9)**Fig. 1.1.9**

- There are four participants Alice, Bob, Eve, and Jane. Assume that Alice has Rs. 100 with here. The public ledger is available to Alice, Bob, Eve, and Jane, which has the initial content/information that Alice has (i.e., Rs. 100).



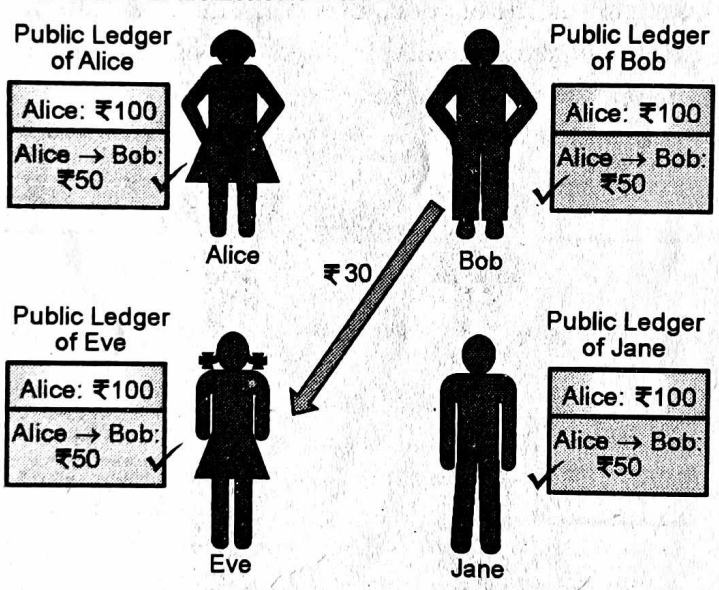
(1A10)**Fig. 1.1.10**

- Now, Alice wants to make a transaction of Rs. 50 to Bob. In this case, another information needs to be updated to all the public ledgers available with Alice, Bob, Eve, and Jane.
- Therefore, this transaction gets updated in all local copies of the public ledger.



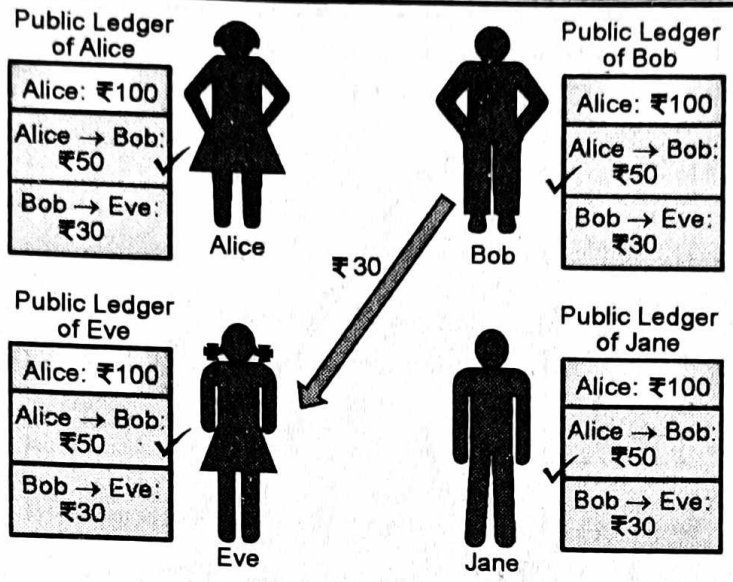
(1A11)**Fig. 1.1.11**

- Now, Bob wants to make a transaction of Rs. 30 to Eve.



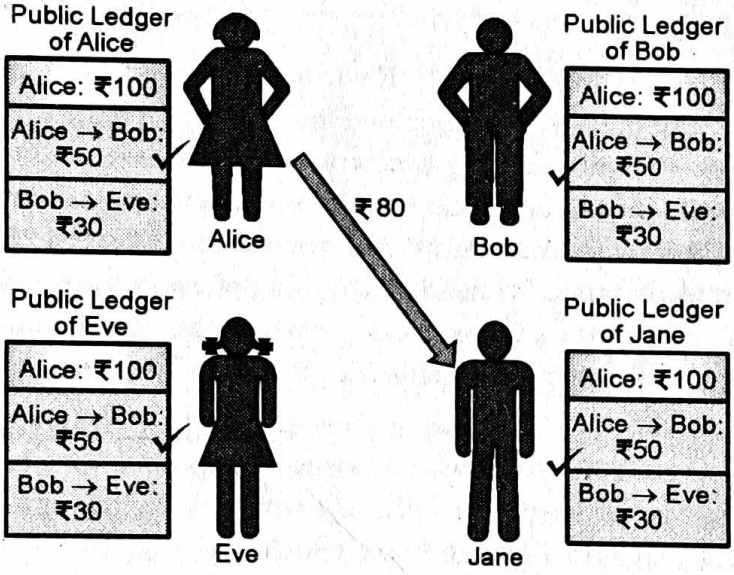
(1A12)**Fig. 1.1.12**

- So, the public ledger of Alice, Bob, Eve, and Jane gets updated with this latest information and this transaction gets updated in all local copies of the public ledger.



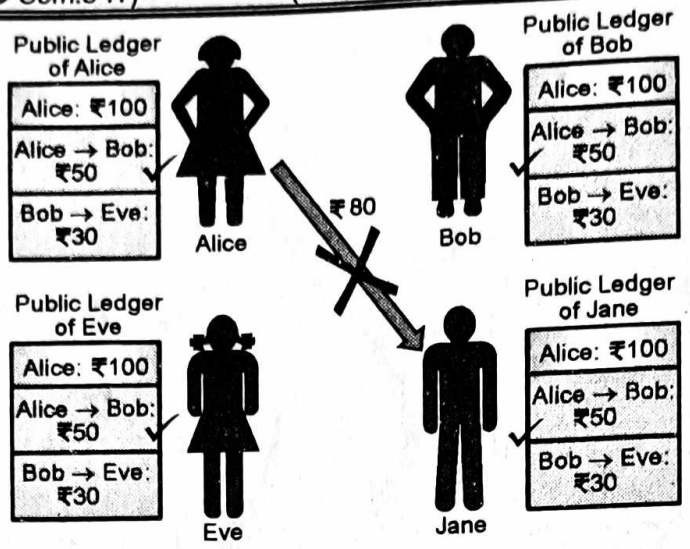
(1A13)**Fig. 1.1.13**

- Now, Alice wants to make a transaction of Rs. 80 to Jane.



(1A14)**Fig. 1.1.14**

- But, from Alice's public ledger, it is evident that she initially had Rs. 100 out of which she gave Rs. 50 to Bob. So now by combining both the transactions, she has only Rs. 50 available with her. If Alice tries to make a transaction of Rs. 80, then this particular transaction is not valid.
- Therefore, all parties will come to know from their public ledger that this transaction is not valid and it is blocked and it cannot be included in their public ledger.



(1A15) **Fig. 1.1.15**

- In this way a public ledger works in case of a decentralized system. Blockchain can be viewed as a public ledger. However, there are numerous aspects that need to be considered.

(1) Protocols for commitment : "Ensure that every valid transaction from the clients are committed and included in the blockchain within a finite time". Whenever someone is making a new transaction, it has to be ensured that this transaction, if it is valid, then it should be committed to existing public ledgers or blockchain otherwise that entry will not be there in the blockchain. So, there should be a mechanism for validity checking of every upcoming transactions from the clients and based on those validity checks, the transaction should be accepted and added in the public ledger/blockchain or deleted or discard such transaction.

(2) Consensus : "Ensure that local copies are consistent and updated." This is an essential aspect in the concept of blockchain. As discussed earlier, there exists a local copy of the information available to every individual parties and there is no such central platform like the bank which will maintain the consistency of the information. Therefore, consensus mechanism ensures that whatever local copy every individual party has, they are consistent/identical with each other.

(3) Security : "Data needs to be tamperproof. Clients may act maliciously or can be compromised." The data that is inserted in a public ledger or blockchain needs to be secure and tamper proof. Because blockchain is distributed to individual parties, everyone is maintaining their local copy of their blockchain. So, it is possible that an individual might make a change in the local copy and broadcast saying that this is the updated information. But the other nodes in the network must be able to understand that whatever this individual is broadcasting is a false/tampered information and the other nodes should not accept such type of information.

(4) **Privacy and Authenticity:** "Data/transactions belong to various clients; and privacy and authenticity needs to be ensured." The data that is present in the blockchain belongs to several clients, and so, it is essential that both privacy and authenticity needs to be safeguarded.

1.1.4 Technical Definition of a Blockchain

A blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way.

Essential keywords

- **Open :** Accessible to all
- **Distributed or Decentralized :** No single party control
- **Efficient :** Fast and scalable
- **Verifiable :** Everyone can check the validity of information
- **Permanent :** Information is persistent

1.2 ELEMENTS OF A BLOCKCHAIN

GQ. State and explain different components or elements of a blockchain.

A variety of blockchain components are available in the market. Some of the major components in a blockchain solution are as follows:

(1) **Ledger :** "It contains the current world state of the ledger and a blockchain of transaction invocations."

Every node in the blockchain network will maintain a ledger of all transactions, and the transactions will maintain the state of the data that is being stored on the blockchain network. In other words, a ledger is a channel's chain and current state data which is maintained by each peer on the channel. The ledger is replicated across all nodes in the network.

(2) **Smart contract :** "It encapsulates business network transactions in code. Transaction invocations result in gets and sets of ledger state." The smart contract is the business logic. An individual can encode his/her own business logic as code/functions and each invocation of this function becomes a transaction on the blockchain. A smart contract is a software running on a ledger to encode assets and the truncation instructions (business logic) for modifying the assets.

(3) **Peer network :** A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block.

(4) **Consensus network :** It is a collection of network data and processing peers forming a blockchain network. It is responsible for maintaining a consistently replicated ledger.

(5) **Membership :** "It manages identity and transaction certificates as well as other aspects of permissioned access." A membership service provides identities for the users to transact on the blockchain.

(6) **Events :** "It creates notifications of significant operations on the blockchain (e.g., a new block) as well as notifications related to smart contracts. It does not include event distribution." Whenever a transaction happens on a blockchain, an individual can create an event notification. So, blockchain will specify that a particular transaction has been committed and will provide the details of the transaction, which can be used to integrate with existing systems of record. Events can also be used to trigger other transactions that might be internal to an organization.

(7) **System management :** The blockchain network is a distributed system that is running across multiple organizations so it requires new ways to create, change, and monitor blockchain components.

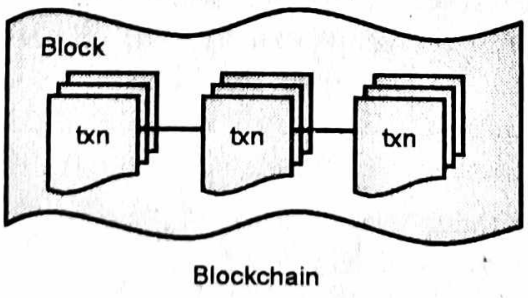
(8) **Wallet :** "It securely manages a user's security credentials." Each user has a digital certificate and is going to be performing transactions using the digital certificate. There needs to be a place where a user can securely store that private information. So, a digital certificate contains the private identity of an individual. He/she should not be sharing the information with anybody else, which is securely managed in a wallet.

(9) **System integration :** It is responsible for integrating blockchain bi-directionally with external systems. It is not a part of the blockchain, but used with it.

☞ **A ledger often consists of the following data structures**

(1) Blockchain

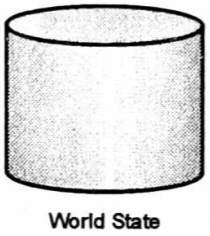
- A linked list of blocks (a hash chain). Each block describes a set of transactions (i.e., the inputs to a smart contract invocation, output, identities, or certificates)
- Hash chaining leads to certain properties of immutability (i.e., blocks cannot be tampered).



(1A20) **Fig. 1.2.1**

(2) World state

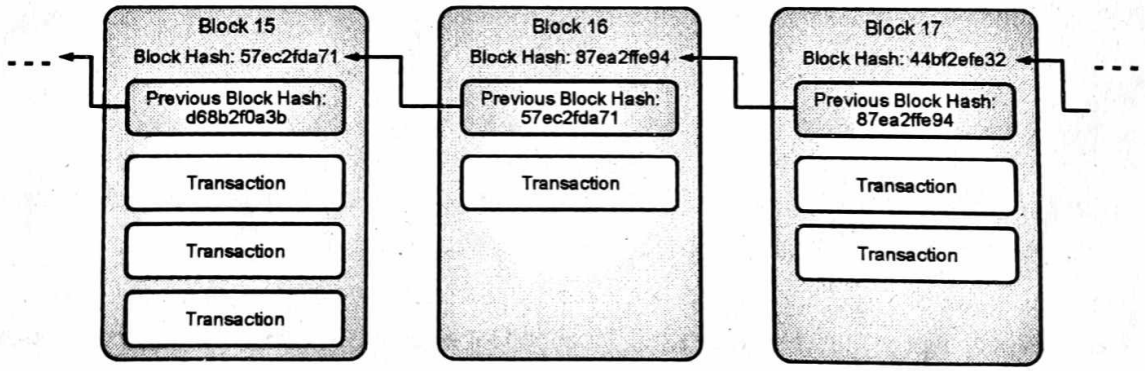
- It is the information that smart contracts can work with.
- It stores the most recent state of smart contracts (*i.e., the output of transactions*)
- In bitcoin, the world state can be thought as the *account balances*.
- It is stored in a traditional database (*e.g., key – value pair*)
- Data elements can be added, modified, deleted, all recorded as transactions on blockchain.



(1A21)**Fig. 1.2.2**

1.2.1 Block in a Blockchain in a Simplified Format

- As can be seen in the Fig. 1.2.3, every block has a hash, which is being chained together. For instance, Block 15 has a hash value **57ec2fda71**, which is getting stored in Block 16. The same process is observed in Block 16 and Block 17.
- A block is a sequence of transactions, i.e., each block can have multiple transactions in them and each transaction can specify which smart contract got invoked or which specific function got invoked and which user invoked the transaction (i.e., some identity of the user will be there in the transaction followed by other details).
- "*Each block can have zero or more transactions and some additional metadata*". Note that the first block that is getting created in the blockchain is called the genesis block, which has special information about the configuration of the network itself, who the participants are, and other configuration information. The genesis block does not have any user transactions, but other blocks in the blockchain can have one or more transactions.

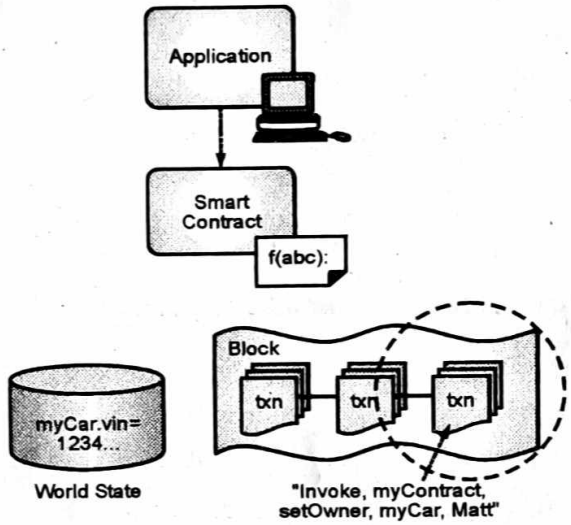


(1A22)**Fig. 1.2.3 : Simplified view of a block in a blockchain**

- Blocks achieve immutability because of the result of a hash function of the previous block and also because of the fact that blocks are getting added through consensus in a decentralized manner. So, not a single organization can unilaterally manipulate blockchain.

1.2.2 Ledger Example : A Change of Ownership Transaction

- The Fig. 1.2.4 shows an example of a transaction and how a ledger might work.
- As it can be seen from the Fig.1.2.4 that there is a smart contract that takes it input for instance an owner (let's say a *car*) and the owner of the car is set to *Matt*.



(1A23)**Fig. 1.2.4 : Example of a transaction & working of a ledger**

 # Transaction input sent from application

```
invoke (myContract, setOwner, myCar, Matt)
...
```

 # Smart contract implementation

```
setOwner(Car, newOwner)
{
    set Car.owner = newOwner
}
```

 # World state: New Contents

```
myCar.vin = 1234
myCar.owner = Matt
myCar.make = Audi
...
```

- From the above code snippet, we have a *setOwner( )* where it sets the owner of a car to a new person, and the new person in this case will be *Matt*.
- The inputs to the *setOwner( )* are **Car** and the **newOwner** of that car. We can see that **Car.owner** is getting modified to **newOwner**, which will be the final output of the transaction.

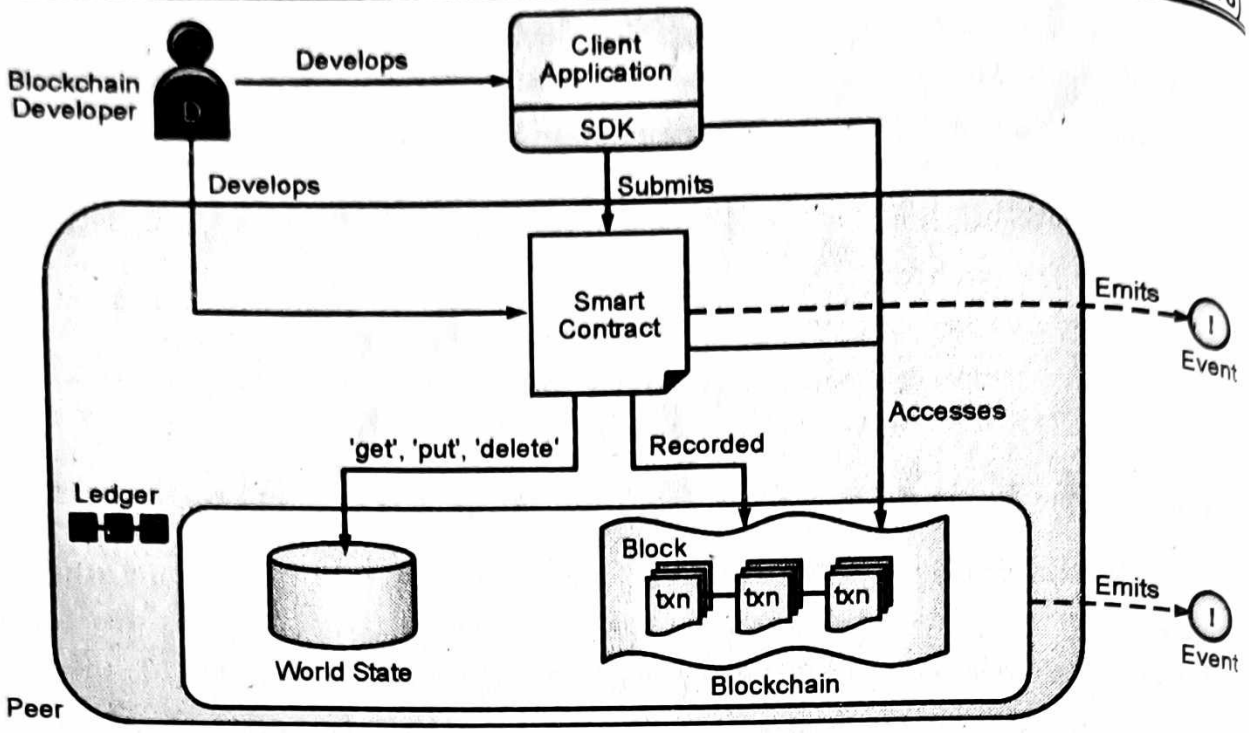
- Now, in the world state, we see how the **myCar** object has been modified *(refer #World state: New Contents)*, which is getting stored in the world state.
- The transaction that contains this information *(dotted circle in the Fig. 1.2.4)* says to invoke **myContract** with *setOwner*, *myCar*, and *Matt* as the arguments. Moreover,

```
myCar . vin = 1234
myCar . owner = Matt
myCar . make = Audi
```

is the output that got written from the contract. All of this gets written as a blockchain transaction in the blockchain as one of the blocks.

1.2.3 How External Applications Interact with the Blockchain Ledger?

- A blockchain developer will develop a client application that will interact with the smart contract. The developer also develops the smart contract. So, this is the business logic that runs inside the blockchain in a decentralized manner.
- In the previous example, **setOwner()** was the part of the smart contract and there is a client application that will invoke functions over a period of time. So, the client application says that Matt is now the new owner of the car and then Matt will be set as the owner. Note that the client application uses an SDK to submit transactions on to the blockchain.
- The client application can also access the blockchain directly to see whether the transactions are actually committed or also look at historic transactions.
- Every transaction that is being invoked gets recorded on the blockchain, and all the elements that are getting modified such *get*, *put*, or *delete* will get recorded on the world state, and whatever is getting modified in the world state will be a part of the transaction. So, this is how a blockchain gets constructed. All this represents a single peer, which means every peer in the network will be performing such a function/task.
- When a client application submit its information to a smart contract, the smart contract gets executed on all the nodes.
- All the nodes simultaneously update their world state and will agree that the output is actually valid and consistent throughout and then the block gets added with the legitimate transaction on to the blockchain, and once a transaction is committed on to the blockchain, it will be called as the final transaction.
- There is an event that gets emitted, as shown in the Fig. 1.2.5, which means that the transaction has been simultaneously committed on all the nodes in the blockchain. The generated events can be used to perform additional processing within an organization.

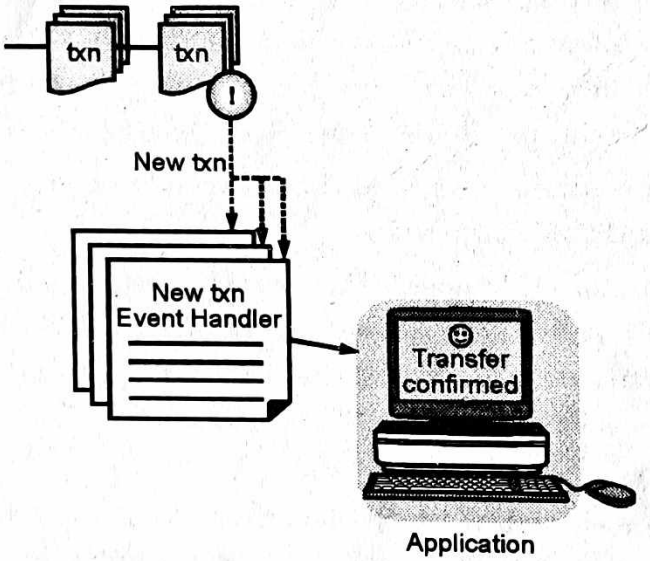


(1A24)**Fig. 1.2.5 : Interaction of an external application with the blockchain ledger**

1.2.4 Blockchain Events

GQ. What are blockchain events?

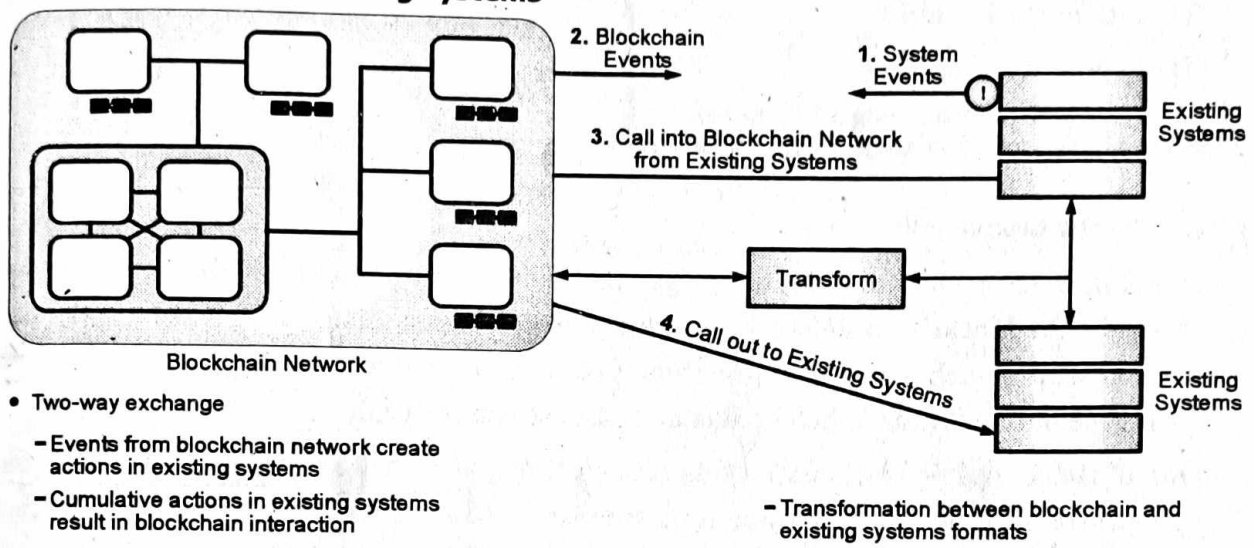
- In computing, an event is an occurrence that can trigger handlers (e.g., disk full, fail transfer completed, mouse clicked, message received, etc.).
- Events are important in asynchronous processing systems like blockchain.



(1A25)**Fig. 1.2.6 : Blockchain events**

- A blockchain can emit events that are useful to application programmers (e.g., transaction has been validated or rejected, block has been added, etc.)
- Events from external systems might also trigger a blockchain activity (e.g., exchange rate has gone down below a threshold value, temperature has risen, time period has elapsed, etc.).

☞ **Integrating with existing systems**



(1A26) **Fig. 1.2.7**

1.3 FEATURES OF BLOCKCHAIN

Blockchain technology has the following main features :

☞ **(1) Decentralization**

- Decentralization in blockchain refers to transferring control and decision making from a centralized entity (individual, organization, or group) to a distributed network.
- Decentralized blockchain networks use transparency to reduce the need for trust among participants. These networks also deter participants from exerting authority or control over one another in ways that degrade the functionality of the network.

☞ **(2) Immutability**

- Immutability means something cannot be changed or altered. No participant can tamper with a transaction once someone has recorded it to the shared ledger.
- If a transaction record includes an error, you must add a new transaction to reverse the mistake, and both transactions are visible to the network.

☞ **(3) Consensus**

- A blockchain system establishes rules about participant consent for recording transactions.

- You can record new transactions only when the majority of participants in the network give their consent.

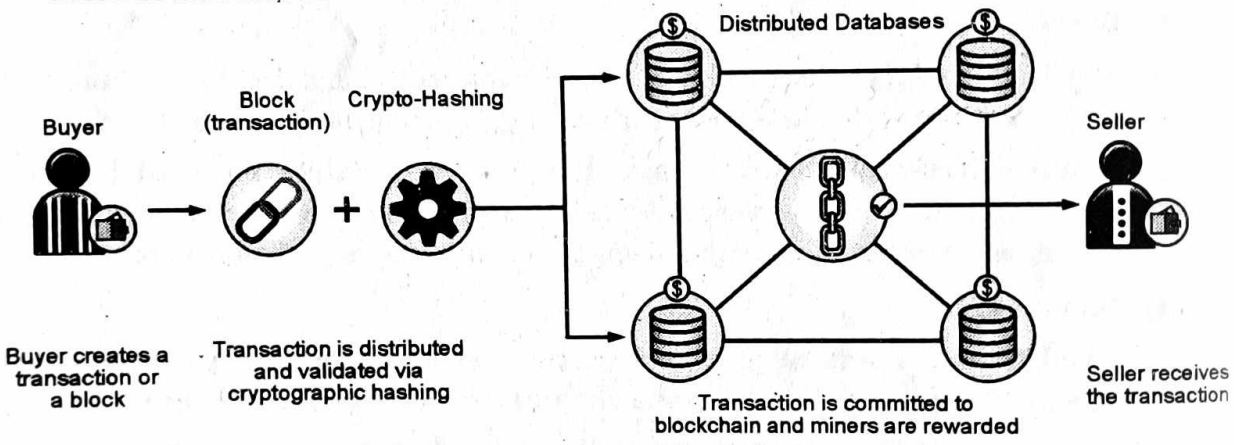
1.4 TYPES OF BLOCKCHAIN

At a glance, there are three major types of blockchain technologies.

(1) Public blockchain
(2) Private blockchain
(3) Consortium/Federated blockchain

(1) Public blockchain

- It is a permission-less distributed ledger technology where any individual can join not only join the blockchain network but also perform transactions. It is a non-restrictive version where each and every peer has a copy of the ledger (i.e., anybody can access such type of blockchain if he/she has an internet connection).
- One of the 1st public blockchains was bitcoin that permitted anybody connected to the internet to carry out transactions in a decentralized manner. Transaction verification takes place through consensus (i.e., proof-of-work (PoW) and proof-of-stake (PoS)).
- Note that the participating nodes need to perform heavy lifting such as validating transactions to make public blockchain work. If a public blockchain does not have the required peers in order to participate and solve transactions, then the blockchain becomes non-functional.



(1A41)**Fig. 1.4.1 : Public blockchain**

Examples : Bitcoin and Ethereum

☞ **Advantages**

(i) Anybody can join public blockchain.

(ii) Everyone feels incentivized to work towards the betterment of a public network.

(iii) It brings transparency to the entire network as the available data is accessible for verification purposes.

(iv) It requires no intermediaries to work.

☞ **Disadvantages**

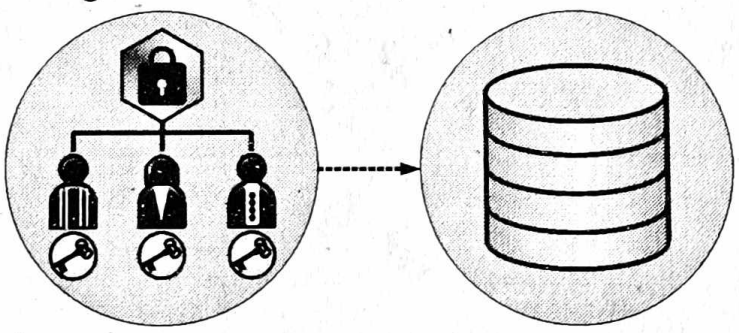
(i) Transaction speed of public blockchain is less (i.e., a transaction could be completed within a few minutes or might take several hours).

(ii) It suffers from scalability (i.e., the more the number of nodes that join the network, the clumsier and slow it becomes).

(iii) It uses the choice of consensus method (e.g., bitcoin uses PoW, which consumes a lot of energy).

▶ **(2) Private blockchain**

- It is type of blockchain that works in a restrictive environment (i.e., in a closed network). It is also a permissioned blockchain that is under the control of an entity.
- It is suitable for organizations who only want selected participants to access a blockchain network. Organizations can set different parameters, such as accessibility and authorization, to the network.
- Private blockchain is similar to a public blockchain, but the only difference is that it allows selected participants to access the network, thereby offering such participants transparency, trust, and security. A private blockchain is usually centralized (i.e., only one authority manages the entire network).



(1A42)**Fig. 1.4.2 : Private blockchain**

Examples : Hyperledger Fabric, Hyperledger Sawtooth, and Corda

☞ **Advantages**

(i) A private blockchain is relatively fast as compared to a public blockchain. This is because there are a few participants in the network, which means that it takes a shorter amount of time for the network to reach consensus resulting in faster transactions.

(ii) They are much more scalable. This is possible because only a few nodes are authorized to validate transactions. For a private blockchain, the network size does not matter. Here, centralization aspect for making a decision is of utmost importance.

☞ **Disadvantages**

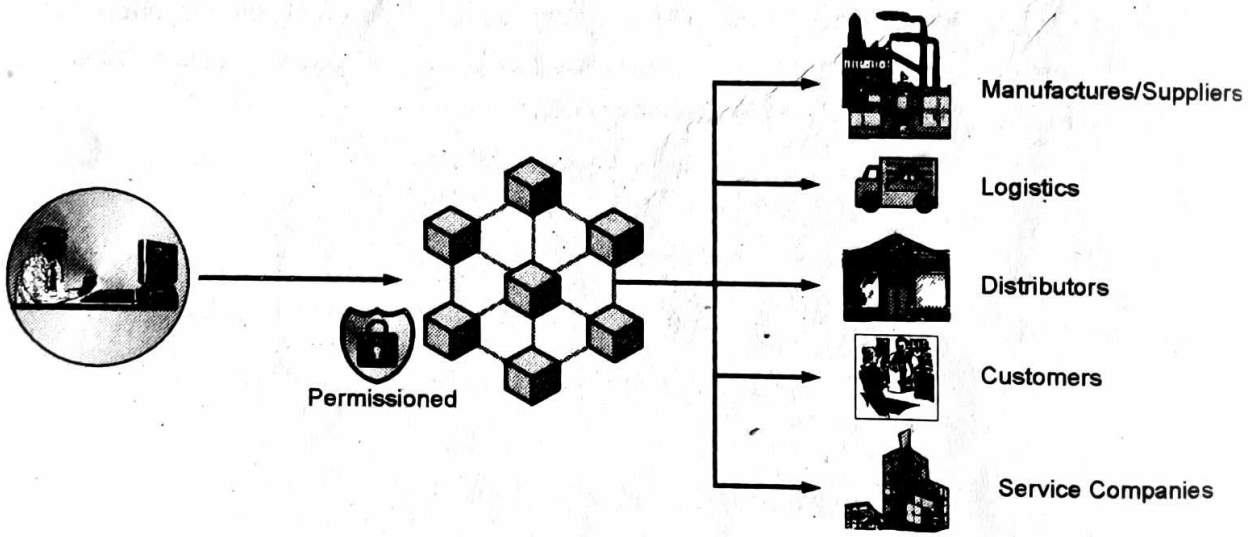
(i) They are not decentralized in a true sense.

(ii) In a private blockchain, achieving trust is difficult because centralized nodes make the last call.

(iii) Security could be an issue in a private blockchain. This could happen when a certain number of nodes compromise the consensus method that is utilized by the private network.

▶ **(3) Consortium/Federated blockchain** Hybrid

- It is a type of blockchain that is preferred by organizations who require both public as well as private blockchain. In a consortium/federated blockchain, certain aspects of an organizations are made public, while others remain private.
- Note that the consensus method in a consortium/federated blockchain is managed by preset nodes. Even though such type of blockchain is not open to mass people, it still holds a decentralized nature.



(1A43) **Fig. 1.4.3 : Consortium/Federated blockchain**

- A consortium/federated blockchain is managed by more than one organizations. As a result, there is no one single force of a centralized outcome. In consortium/federated blockchain, there exists a validator node that not only validates transactions but also initiates or receives transactions.
- The member nodes on the other hand can only receive or initiate transactions. Overall, a consortium/federated blockchain offers all the features of a private blockchain, including transparency, privacy, and efficiency, without a single party having the entire power.

Examples : Marco Polo and IBM Food Trust

☞ **Advantages**

(i) Better customizability and control over resources.

(ii) Better scalability and offers access controls.

(iii) More secure and efficient.

☞ **Disadvantages**

(i) Even though a consortium/federated blockchain is secure, the entire network can be compromised due to a member's integrity.

(ii) Less transparency

(iii) Censorship and enforced regulations might adversely affect network functionality.

(iv) Less anonymous compared to public and private blockchain.

☞ **Comparison between different types of blockchain**

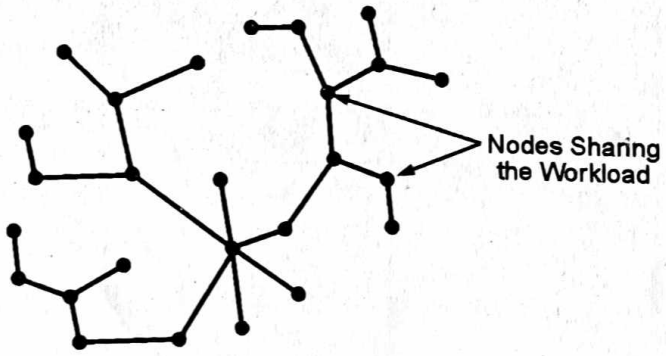
Sr. No.	Parameters	Public Blockchain	Private Blockchain	Consortium/Federated Blockchain
1.	Definition	It is open to everyone and anyone can participate.	It is controlled by owners and access is limited to certain users.	It is a combination of public and private blockchain, which means some processes are kept private and others public.
2.	Cost of transactions	Costly	Not so costly	Not so costly
3.	KYC needed	No	Yes	Yes
4.	Incentive	Public blockchain incentivizes participants to grow the network.	Private blockchain is limited; hence, it has no similar incentives as that of a public blockchain.	Consortium/Federated blockchain can opt to incentivize users if they wish.

Sr. No.	Parameters	Public Blockchain	Private Blockchain	Consortium/Federated Blockchain
5.	Transparency	It is entirely transparent	It is transparent to only those users who are granted access	Transparency depends on how owners set rules
6.	Transaction speed	Slow	Faster than public blockchain	Fast

1.5 WHAT IS DISTRIBUTED LEDGER TECHNOLOGY?

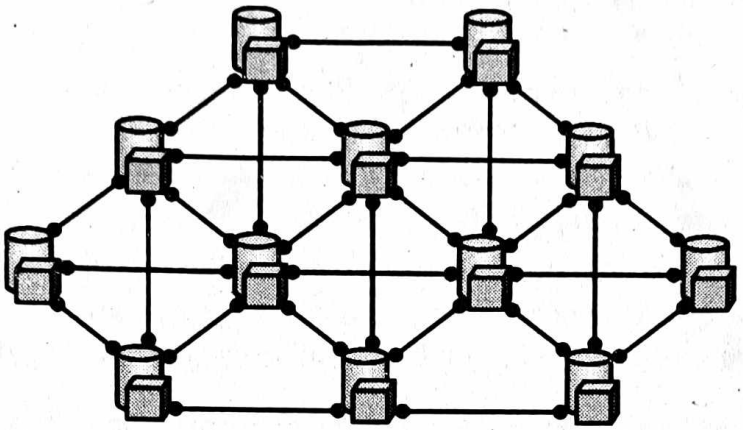
Distributed Ledger Technology

- A distributed ledger is a decentralized ledger of all the information that is recorded on the blockchain by consensus. In other words, a distributed ledger technology is defined as *a decentralized database that can securely record and share financial, physical, or electronic access across a geographical network through transparent updates of information.*
- A distributed ledger can be considered to be a database similar to an accounting ledger where financial transactions are recorded, except that it is not recorded to financial data. Herein, a transaction refers to any digital data such as text, images, or audio files.
- The database can be distributed to all locations and can be accessed by every single member present in the blockchain network, thereby ensuring incorruptibility as malicious changes cannot be made when each node has simultaneous access to all records.
- Note that a decentralized system does not be a completely distributed one. The below Fig. 1.5.1 represents a decentralized network where processing work is shared with sub-nodes.



(1A39)**Fig. 1.5.1**

- Such decentralization is different from the one that is observed in blockchain. In DLT of blockchain, a database copy is available on every user's computer. This database is independent of a central authority and any changes to the ledger have to be agreed upon by all nodes.
- Such a mechanism is called as the consensus mechanism. Once consensus is attained, the database is updated to all nodes in the network. At any given point in time, information is synchronized across all nodes; so, it is referred to as distributed consensus.



(1A40)**Fig. 1.5.2 : Distributed Ledger Technology**

☞ **Benefits of distributed ledger technology**

(1) **Transparent and secure :** As data is shared and visible to all nodes, it is not easy to make any unauthorized changes. Every node that participates in the network maintains a copy of the ledger, thereby preventing a single point of failure. Any entry has to be agreed upon by all parties making a distributed ledger secure and tamper proof.

(2) **Decentralized :** Unlike a centralized system, every node has control over his/her data. Decentralization offers users the power to take appropriate decisions as to what will be in his/her data record. Consensus methods help in securely adding appropriate data in a distributed ledger, thereby offering innate trust in a system.

(3) **Efficient :** With trustless and distributed nature of blockchain's DLT, the effort of capturing, validating, and synchronizing individual sets of information by mediators can be eliminated, thereby reducing the chances of human error and improvement of operational efficiency.

(4) **Cost savings :** Disintermediated systems can save on additional/bottom-line costs while realizing near-time transactions and efficiencies.

1.6 DLT V/S BLOCKCHAIN

- A common misconception is that blockchain and distributed ledger technology are one and the same. The former (blockchain) is actually a type of the latter (DLT) which, because of its popularity, has become ingrained in people's minds as what the product is.
- Think of blockchain and distributed ledger in the same way one might think of Post-it and sticky notes. In the same way not all sticky notes are Post-it, not all DLTs are blockchains – but all blockchains are DLTs.
- The transactional architecture of blockchain bundles validated transactions into a sequential chain of time-stamped, cryptographically hashed blocks that are globally available to all parties on the network. By contrast, transactions on a standard DLT ledger are only available to included parties and data does not need to be structured in blocks or require a sequential chain.
- DLTs may also simply be a distributed system of record (SoR), without facilitating financial exchanges through a form of cryptocurrency as is common with a blockchain. The similarities and differences of blockchain versus DLT are summarised in the table below.

	Blockchain	DLT
Decentralized	Yes	Yes
Immutable	Yes	Yes
Distributed	Yes, across anyone running a node	Yes, but not everyone has all records
Consensus	Same throughout network	Pluggable at transaction level
Permissions	Open and permissionless	Closed and permissioned
KYC	Not required	Required for all participants
Blocks	Yes	Optional
Cryptocurrency	Maybe	Optional
Tokens	Maybe	Optional
Scalability	Lower	Higher

1.7 CAP THEOREM

CAP theorem stands for Consistency, Availability, and Partition Tolerance. According to the theorem, a distributed system cannot always ensure consistency, availability, and partition tolerance.

When things go wrong, we must prioritize at most two distributed system features and trade-offs between them.

(1) Availability

- Availability means that all clients who request data receive a response even if one or more nodes are down. In a distributed system, every operational node replies to each valid request made to it, to put it another way.
- **Example :** Imagine you are the customer of a well-known vehicle company in your city because of the incredible deals and services it provides. In addition, they provide fantastic customer service, so you can contact them whenever you have questions or issues and get answers right away. The car company is able to connect every consumer who phones to one of its customer service representatives. Any information needed by the customer regarding his cars, such as the service date, the insurance plan, or other details, can be obtained. Because any customer can connect to the business or its operator and obtain information about the user or client, we refer to this as availability.

(2) Consistency

- Consistency means that the duplicated data item will appear in the same copies on all nodes during different transactions. an assurance that each node in a distributed cluster returns the same, most recent, and successful writer.
- Every client's perception of the data must be consistent to be considered consistent. Sequential consistency, which is a particularly powerful type of consistency, is referred to as consistency in CAP.
- **Example :** Recently the insurance policy of your car got outdated and you want to update or get a new insurance policy for your car. You decide to call the bank or the insurance company and update it with them. When you call, you connect with an agent. This agent asks you for the relevant details of your previous policy. But once you have put down the phone, you realize that you missed one detail. So you frantically call the agent again. But, this time when you call, you connect with a different agent but then also, they are able to access your records as well and know that you are registering for your new insurance policy. They make the relevant changes in the house number and the rest of the address is the same as the one you told the last operator. We call this Consistency because even though you connect to a different customer care operator, they were able to retrieve the same information.

(3) Partition Tolerance

- A communication breakdown a momentary delay or lost connection between nodes is referred to as a partition in a distributed system. Partition tolerance describes the ability of a cluster to function even in the face of numerous communication failures between system nodes.
- **Example :** Unfortunately, you need to sell your car because it's old and outdated and you don't use it very often. So you list all of the specifics about your automobile on a website for sale, and then you get in touch with a buyer. He starts negotiating because he wants to acquire your car and hence wants to complete the agreement process. However, because the bargaining was not mutual, neither of you could sign the agreement. Therefore, we might conclude that the agreement has been breached or that there is no partition tolerance in this situation.
- In a distributed system, the processes can only be based on two of the three CAP theorem properties. These are to be, one of CA, CP or AP at a go.
  - **CA :** CA database provides availability and consistency among all the nodes. However, it cannot accomplish this if there is a partition between any two system nodes, hence it is unable to provide fault tolerance. For example, applications used in banking and finance demand available and consistent data.
  - **AP :** AP database means that the system continues to operate even in the presence of node failures. AP-based systems compromise consistency and availability. Non- distributed databases like PostgreSQL uses AP-based database systems.
  - **CP :** CP database means that the system continues to operate even though network failures are occurring in the database. CP systems are strongly consistent but they are not properly available.
- CAP Theorem or Brewer's theorem states that it is feasible to provide either consistency or availability but not both in the event of a network failure on a distributed database, a theory from theoretical computer science about distributed data stores. In other words, according to the CAP theorem, a distributed database system that experiences a partition must choose between Consistency and Availability.
- We must simultaneously communicate over the network and store data among several nodes in a distributed system. A distributed system frequently falls victim to network failures because of its reliance on network calls in a significant way. Tolerance for partitions is crucial. In this situation, we must decide, based on our needs, whether to prioritize consistency or availability.

- With blockchain technology, immediate consistency is frequently sacrificed for availability and partition tolerance. By requiring a specific amount of "confirmations," blockchain consensus techniques are simply simplified to eventual consistency.
- Network failures can affect any distributed system, hence network partitioning is usually required. There are just two choices remaining in the event of a partition: consistency or availability. The system will return an error or time out if a specific piece of information cannot be guaranteed to be current owing to network segmentation when consistency is chosen above availability. The system will always process the query and attempt to return the most recent version of the data even if it cannot ensure that it is up to date because of network partitioning when availability is preferred over consistency.
- Blockchain is a decentralized database that manages a shared ledger that cannot be altered. Transactions are what make up the shared ledger. Consensus techniques are used to record transactions inside the shared ledger. Sharing distributed transactions naturally raises questions about the CAP theorem.
- Consistency is sacrificed in Blockchain due to the priority given to Availability and Partition Tolerance. In this case, Partition tolerance (P), Availability (A), and Consistency (C) on the blockchain are not attained simultaneously; instead, they are acquired over time.

1.8 BYZANTINE GENERALS PROBLEM

- In 1982, Byzantine Generals Problem was invented by Leslie Lamport, Robert Shostak, and Marshall Pease.
- Byzantine Generals Problem is an impossibility result which means that the solution to this problem has not been found yet as well as helps us to understand the importance of blockchain.
- It is basically a game theory problem that provides a description of the extent to which decentralized parties experience difficulties in reaching consensus without any trusted central parties.
- Byzantine army is divided into many battalions in this classic problem called Byzantine Generals problem, with each division led by a general.
- The generals connect via messenger in order to agree to a joint plan of action in which all battalions coordinate and attack from all sides in order to achieve success. It is probable that traitors will try to sabotage their plan by intercepting or changing the messages. As a result, the purpose of this challenge is for all of the faithful commanders to reach an agreement without the imposters tampering with their plans.

Byzantine Generals Problem

- Money is one such commodity whose value should be same throughout the society, that is everyone should agree upon the value of a certain amount of money, despite all the differences therefore in the initial times, precious metals and rare goods were chosen as money because their value was seen equally throughout the society, but in some cases such as precious metals the purity of the metals could not be known for sure or checking the purity was an extremely tedious task which turned out to be very inefficient for the daily transactions, therefore it was decided upon to replace gold with a central party which would be highly trustable chosen by the people in the society to establish and maintain the system of money.
- But with time it was later realized that those central parties, how much-ever qualified were still not completely trustworthy as it was so simple for them to manipulate the data.
- Centralized systems do not address Byzantine Generals problem, which requires that truth be verified in an explicitly transparent way, yet centralized systems give no transparency, increasing the likelihood of data corruption. They forgo transparency in order to attain efficiency easily and prefer to avoid dealing with the issue entirely.
- The fundamental issue of centralized systems, however, is that they are open to corruption by the central authority, which implies that the data can be manipulated by anyone who has control of the database itself because the centralized system concentrates all power on one central decision maker.
- Therefore, Bitcoin was invented to make the system of money decentralized using blockchain to make money verifiable, counterfeit-resistant, trustless, and separate from a central agency.

Byzantine Fault Tolerance (BFT)

- Byzantine Fault Tolerance was developed as an inspiration in order to address Byzantine Generals Problem. Byzantine Generals Problem, a logical thought experiment where multiple generals must attack a city, is where the idea for BFT originated. Byzantine Fault Tolerance is one of the core characteristics of developing trustworthy blockchain rules or features is tolerance.
- When two-thirds of the network can agree or reach a consensus and the system still continues to operate properly, it is said to have BFT. Blockchain networks' most popular consensus protocols, such as proof-of-work, proof-of-stake, and proof-of-authority, all have some BFT characteristics.
- In order to create a decentralized network, BFT is essential.

1.9 CONSENSUS MECHANISM

- The consensus method determines precise network structure. For instance, BFT has a leader as well as peers who can and cannot validate.
- In order to maintain the sequence of blockchain transactions and the consistency of global state through local transaction replay, consensus messages must pass between relevant peers.
- More inventive approaches to designing BFT systems will be found and put into practice as more individuals and companies investigate distributed and decentralized systems.
- Systems that use BFT are also employed in sectors outside of blockchain, such as nuclear power, space exploration, and aviation. Proof-of-work in blockchain for Byzantine Generals Problem.

Byzantine Generals Problem in a Distributed System

- In order to address this issue, honest nodes (such as computers or other physical devices) must be able to establish an agreement in the presence of dishonest nodes.
- In Byzantine agreement issue, an arbitrary processor initializes a single value that must be agreed upon, and all nonfaulty processes must agree on that value. Every processor has its own beginning value in the consensus issue, and all nonfaulty processors must agree on a single common value. Byzantine army's position can be seen in computer networks.
- The divisions can be viewed as computer nodes in the network, and the commanders as programs running a ledger that records transactions and events in the order that they occur.
- The ledgers are the same for all systems, and if any of them is changed, the other ledgers are updated as well if the changes are shown to be true, so all distributed ledgers should be in agreement.

1.10 CRYPTOGRAPHIC PRIMITIVES AND ITS TYPES

Cryptographic Primitives

- In blockchain, there are no third parties or government involved. It is completely decentralized and various transactions happen in these networks. So, security is of utmost importance in blockchain. Cryptographic primitives are used for building cryptographic protocols for a strong secured network.

- They are the low-level algorithms that are used to build algorithms. They are the basic building blocks of the cryptosystem. The programmers develop new cryptographic algorithms with the help of cryptographic primitives.
- Cryptographic primitives are the basic building blocks for the development of security protocols. Hence, they are an integral part of blockchain because of the following reasons :

  (1) **Security :** To secure a transaction in the network or confidential information, strong cryptography is required. So cryptographic primitives are used to develop high-level algorithms.

  (2) **Encryption and Decryption :** The Cryptographic primitives are used to develop encryption and decryption algorithms. Encryption algorithms encrypt the data and decryption algorithms decrypt the data as and when required.

  (3) **Validation :** The validation of data is done with the help of digital signatures. These digital signatures are public key primitives which the receivers use to validate the message.

  (4) **Specific :** Cryptographic primitives are very specific in nature. It means one cryptographic primitive can perform only one function. For example, The encryption algorithms developed using crypto primitives are only responsible for encrypting the text. It is not responsible for hashing or decryption.

Types of Cryptographic Primitives

(1) **One-way Hash Functions :** It is a mathematical function used to encrypt variable length data to fixed binary data. It is a one-way function. It means that once the input has been converted to a binary sequence, there is no scope for reverting back. It is also known as fingerprint or compression function. It is to be noted that a slight change in input can also change the hash function. This is known as the avalanche effect. A popular hash function is SHA-256.

(2) **Symmetric Key Cryptography :** This is also known as Symmetric Encryption. Suppose a message is encrypted using a key. The message is now converted to ciphertext which is readable but has no meaning. The same key is used to decrypt the message. A key is a variable used to encrypt or decrypt a text. It is basically used to 'lock' or 'unlock' data. In this cryptography, the key is shared between two users. The sharing of keys is a problem. However, this technique is faster than public-key cryptography. Examples are Advanced Encryption Standard (AES) and the Data Encryption Standard (DES).

(3) **Asymmetric Key Cryptography** : It is also known as public key cryptography. Since there is a problem with sharing keys in symmetric encryption, this method is used. Here one key is public and another key is private. The public key is used to encrypt or 'lock' data. The private key is only accessible to the receiver. The receiver

uses a private key to 'unlock' the data. For example, Suppose Bob encrypts the data using the public key. The public key is available to everyone but this key works in one way. The receiver has the private key which works in one way and is used to decrypt the message. Examples of public key algorithms are DSA and RSA.

(4) **Randomized Algorithms :** These algorithms produce random ciphertexts for encryption. The ciphertext is an encrypted text. It is very secure as random texts are produced for encryption. It is impossible for hackers to find various combinations of texts. It employs randomness as a logical part. It uses random inputs and gives correct output. For Example, Monte Carlo

(5) **Mix Network :** It is a routing algorithm that uses public key cryptography to encrypt data. The proxy servers take messages, encrypt them and shuffle them so that communication cannot be traced. It basically breaks the flow of messages between the sender and the target.

(6) **Retrieval of Private Information :** It is a protocol that allows the user to retrieve information from the database. Other users don't get to know about it. The user can anonymously retrieve data without taking permission.

(7) **Initialization Vector :** It is a number that is used along with a key for encryption. It is used to prevent the duplicate generation of ciphertext.

1.11 DATA STRUCTURE USED IN BLOCKCHAIN

- The blockchain data structure is an ordered, back-linked list of blocks of transactions. The blockchain can be stored as a flat file, or in a simple database.
- The Bitcoin Core client stores the blockchain metadata using Google's LevelDB. Blocks are linked "back," each referring to the previous block in the chain. The blockchain is often visualized as a vertical stack, with blocks layered on top of each other and the first block serving as the foundation of the stack.
- The visualization of blocks stacked on top of each other results in the use of terms such as "height" to refer to the distance from the first block, and "top" or "tip" to refer to the most recently added block.
- Each block within the blockchain is identified by a hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block also references a previous block, known as the parent block, through the "previous block hash" field in the block header.
- In other words, each block contains the hash of its parent inside its own header. The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the genesis block.

- Although a block has just one parent, it can temporarily have multiple children. Each of the children refers to the same block as its parent and contains the same (parent) hash in the "previous block hash" field.
- Multiple children arise during a blockchain "fork," a temporary situation that occurs when different blocks are discovered almost simultaneously by different miners (see Blockchain Forks).
- Eventually, only one child block becomes part of the blockchain and the "fork" is resolved. Even though a block may have more than one child, each block can have only one parent. This is because a block has one single "previous block hash" field referencing its single parent.
- The "previous block hash" field is inside the block header and thereby affects the current block's hash. The child's own identity changes if the parent's identity changes.
- When the parent is modified in any way, the parent's hash changes. The parent's changed hash necessitates a change in the "previous block hash" pointer of the child. This in turn causes the child's hash to change, which requires a change in the pointer of the grandchild, which in turn changes the grandchild, and so on.
- This cascade effect ensures that once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks. Because such a recalculation would require enormous computation, the existence of a long chain of blocks makes the blockchain's deep history immutable, which is a key feature of bitcoin's security.
- One way to think about the blockchain is like layers in a geological formation, or glacier core sample.
- The surface layers might change with the seasons, or even be blown away before they have time to settle. But once you go a few inches deep, geological layers become more and more stable.
- By the time you look a few hundred feet down, you are looking at a snapshot of the past that has remained undisturbed for millions of years. In the blockchain, the most recent few blocks might be revised if there is a chain recalculation due to a fork. The top six blocks are like a few inches of topsoil.
- But once you go more deeply into the blockchain, beyond six blocks, blocks are less and less likely to change. After 100 blocks back there is so much stability that the coinbase transaction containing newly mined bitcoins can be spent.
- A few thousand blocks back (a month) and the blockchain is settled history, for all practical purposes. While the protocol always allows a chain to be undone by a longer chain and while the possibility of any block being reversed always exists, the probability of such an event decreases as time passes until it becomes infinitesimal.

1.11.1 Structure of a Block

- A block is a container data structure that aggregates transactions for inclusion in the public ledger, the blockchain. The block is made of a header, containing metadata, followed by a long list of transactions that make up the bulk of its size.
- The block header is 80 bytes, whereas the average transaction is at least 250 bytes and the average block contains more than 500 transactions. A complete block, with all transactions, is therefore 1,000 times larger than the block header. Table 1.11.1 describes the structure of a block.

Table 1.11.1 : Structure of a block

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

1.11.2 Block Header

- The block header consists of three sets of block metadata. First, there is a reference to a previous block hash, which connects this block to the previous block in the blockchain.
- The second set of metadata, namely the difficulty, timestamp, and nonce, relate to the mining competition.
- Table 1.11.2 describes the structure of a block header.

Table 1.11.2 : Structure of the block header

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the Merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

1.11.3 Block Header Hash and Block Height

- The primary identifier of a block is its cryptographic hash, a digital fingerprint, made by hashing the block header twice through the SHA256 algorithm. The resulting 32-byte hash is called the block hash but is more accurately the block header hash because only the block header is used to compute it.
- For example,
  000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f is the block hash of the first bitcoin block ever created.
- The block hash identifies a block uniquely and unambiguously and can be independently derived by any node by simply hashing the block header.
- Note that the block hash is not actually included inside the block's data structure neither when the block is transmitted on the network, nor when it is stored on a node's persistence storage as part of the blockchain.
- Instead, the block's hash is computed by each node as the block is received from the network. The block hash might be stored in a separate database table as part of the block's metadata, to facilitate indexing and faster retrieval of blocks from disk.
- A second way to identify a block is by its position in the blockchain, called the block height. The first block ever created is at block height 0 (zero) and is the same block that was previously referenced by the following block hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f.
- A block can thus be identified two ways: by referencing the block hash or by referencing the block height. Each subsequent block added "on top" of that first block is one position "higher" in the blockchain, like boxes stacked one on top of the other. The block height on January 1, 2014, was approximately 278,000, meaning there were 278,000 blocks stacked on top of the first block created in January 2009.
- Unlike the block hash, the block height is not a unique identifier. Although a single block will always have a specific and invariant block height, the reverse is not true– the block height does not always identify a single block.
- Two or more blocks might have the same block height, competing for the same position in the blockchain.
- This scenario is discussed in detail in the section Blockchain Forks. The block height is also not a part of the block's data structure; it is not stored within the block.
- Each node dynamically identifies a block's position (height) in the blockchain when it is received from the bitcoin network. The block height might also be stored as metadata in an indexed database table for faster retrieval.

1.11.4 The Genesis Block

- The first block in the blockchain is called the genesis block and was created in 2009. It is the common ancestor of all the blocks in the blockchain, meaning that if you start at any block and follow the chain backward in time, you will eventually arrive at the genesis block.
- Every node always starts with a blockchain of at least one block because the genesis block is statically encoded within the bitcoin client software, such that it cannot be altered. Every node always "knows" the genesis block's hash and structure, the fixed time it was created, and even the single transaction within. Thus, every node has the starting point for the blockchain, a secure "root" from which to build a trusted blockchain.
- The statically encoded genesis block inside the Bitcoin Core client, in chainparams.cpp. The following identifier hash belongs to the genesis block :

  000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
- Using the Bitcoin Core reference client on the command line :

```
$ bitcoindgetblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
{
    "hash":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations":308321,
    "size":285,
    "height":0,
    "version":1,
    "merkleroot":"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx":[
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
],
"time":1231006505,
"nonce":2083236893,
"bits":"1d00ffff",
"difficulty":1.00000000,
"nextblockhash":"00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

- The genesis block contains a hidden message within it. The coinbase transaction input contains the text "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks." This message was intended to offer proof of the earliest date this block was created, by referencing the headline of the British newspaper *The Times*.
- It also serves as a tongue-in-cheek reminder of the importance of an independent monetary system, with bitcoin's launch occurring at the same time as an unprecedented worldwide monetary crisis. The message was embedded in the first block by Satoshi Nakamoto, bitcoin's creator.

1.11.5 Linking Blocks in Blockchain

- Bitcoin full nodes maintain a local copy of the blockchain, starting at the genesis block. The local copy of the blockchain is constantly updated as new blocks are found and used to extend the chain.
- As a node receives incoming blocks from the network, it will validate these blocks and then link them to the existing blockchain. To establish a link, a node will examine the incoming block header and look for the "previous block hash."
- Let's assume, for example, that a node has 277,314 blocks in the local copy of the blockchain. The last block the node knows about is block 277,314, with a block header hash of 00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.
- The bitcoin node then receives a new block from the network, which it parses as follows :

```
{
"size":43560,
"version":2,
"previousblockhash":
"00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
"merkleroot":
"5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
"time":1388185038,
"difficulty":1180923195.25802612,
"nonce":4215469401,
"tx":[
"257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",

#[...manymoretransactionsomitted...]

"05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
]
}
```

- At this new block, the node finds the previous block hash field, which contains the hash of its parent block. It is a hash known to the node, that of the last block on the chain at height 277,314.
- Therefore, this new block is a child of the last block on the chain and extends the existing blockchain. The node adds this new block to the end of the chain, making the blockchain longer with a new height of 277,315.

- Fig. 1.11.1 shows the chain of three blocks, linked by references in the previous block hash field.

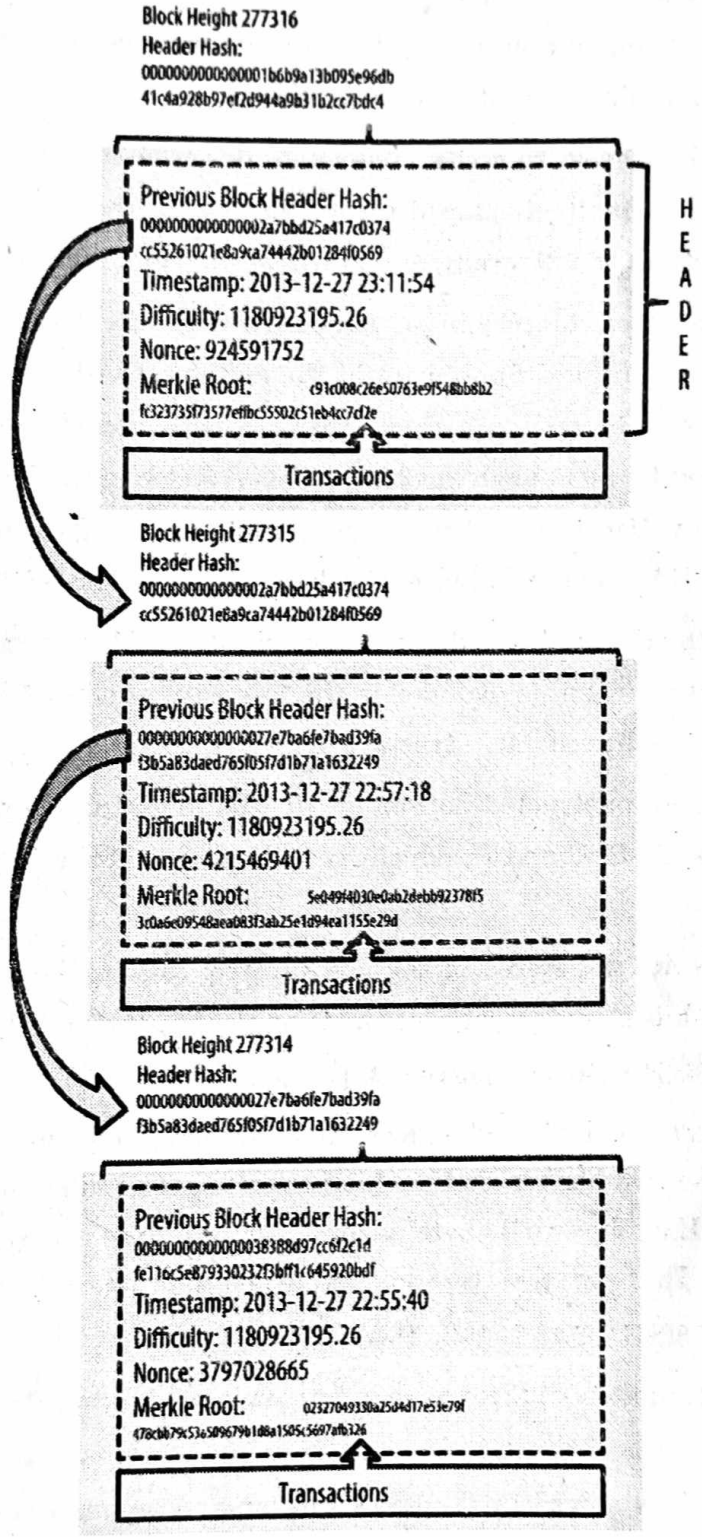


Fig. 1.11.1 : Blocks linked in a chain by reference to the previous block header hash

1.11.6 Merkle Tree

- A Merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data. Merkle trees are binary trees containing cryptographic hashes.
- The term "tree" is used in computer science to describe a branching data structure, but these trees are usually displayed upside down with the "root" at the top and the "leaves" at the bottom of a diagram, as you will see in the examples that follow.
- Merkle trees are used in bitcoin to summarize all the transactions in a block, producing an overall digital fingerprint of the entire set of transactions, providing a very efficient process to verify whether a transaction is included in a block. A Merkle tree is constructed by recursively hashing pairs of nodes until there is only one hash, called the *root*, or *Merkle root*. The cryptographic hash algorithm used in bitcoin's Merkle trees is SHA256 applied twice, also known as double-SHA256.
- When N data elements are hashed and summarized in a Merkle tree, you can check to see if any one data element is included in the tree with at most $2*\log_2(N)$ calculations, making this a very efficient data structure.
- The Merkle tree is constructed bottom-up. In the following example, we start with four transactions, A, B, C and D, which form the *leaves* of the Merkle tree, as shown in 5.
- The transactions are not stored in the Merkle tree; rather, their data is hashed and the resulting hash is stored in each leaf node as H_A, H_B, H_C, and H_D:

  H~A~ = SHA256(SHA256(Transaction A))
- Consecutive pairs of leaf nodes are then summarized in a parent node, by concatenating the two hashes and hashing them together. For example, to construct the parent node H_{AB}, the two 32-byte hashes of the children are concatenated to create a 64-byte string. That string is then double-hashed to produce the parent node's hash:

  H~AB~ = SHA256(SHA256(H~A~ + H~B~))
- The process continues until there is only one node at the top, the node known as the Merkle root.
- That 32-byte hash is stored in the block header and summarizes all the data in all four transactions.

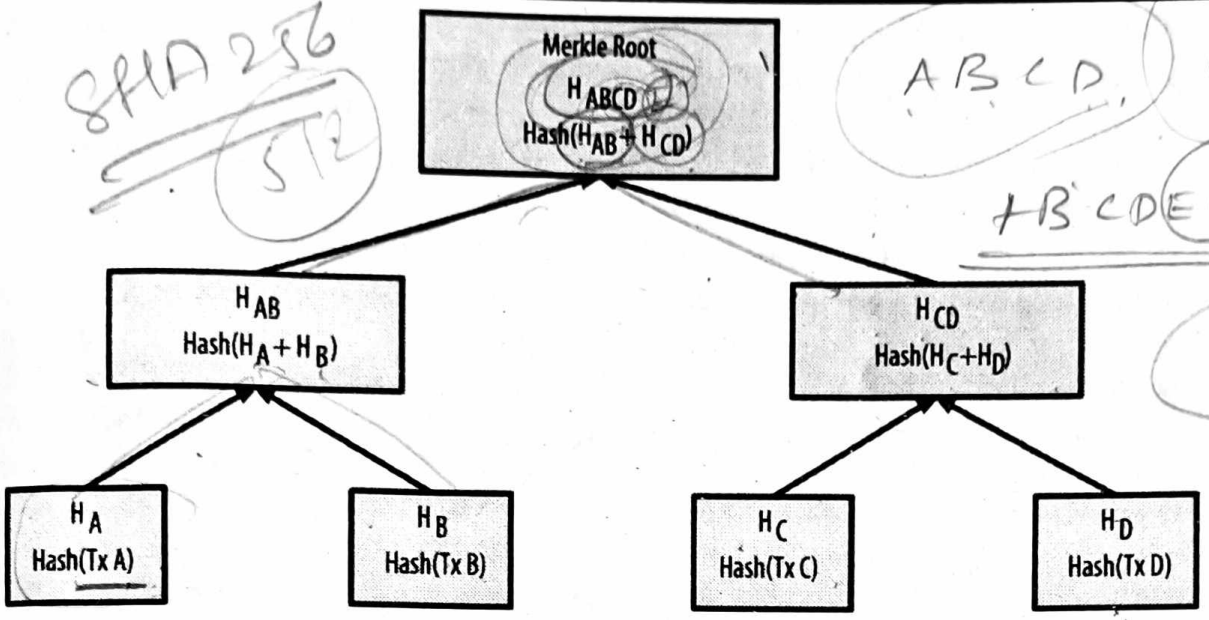


Fig. 1.11.2 : Calculating the nodes in a Merkle tree

- Because the Merkle tree is a binary tree, it needs an even number of leaf nodes. If there is an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes, also known as a *balanced tree*. This is shown in Fig. 1.11.3, where transaction C is duplicated.

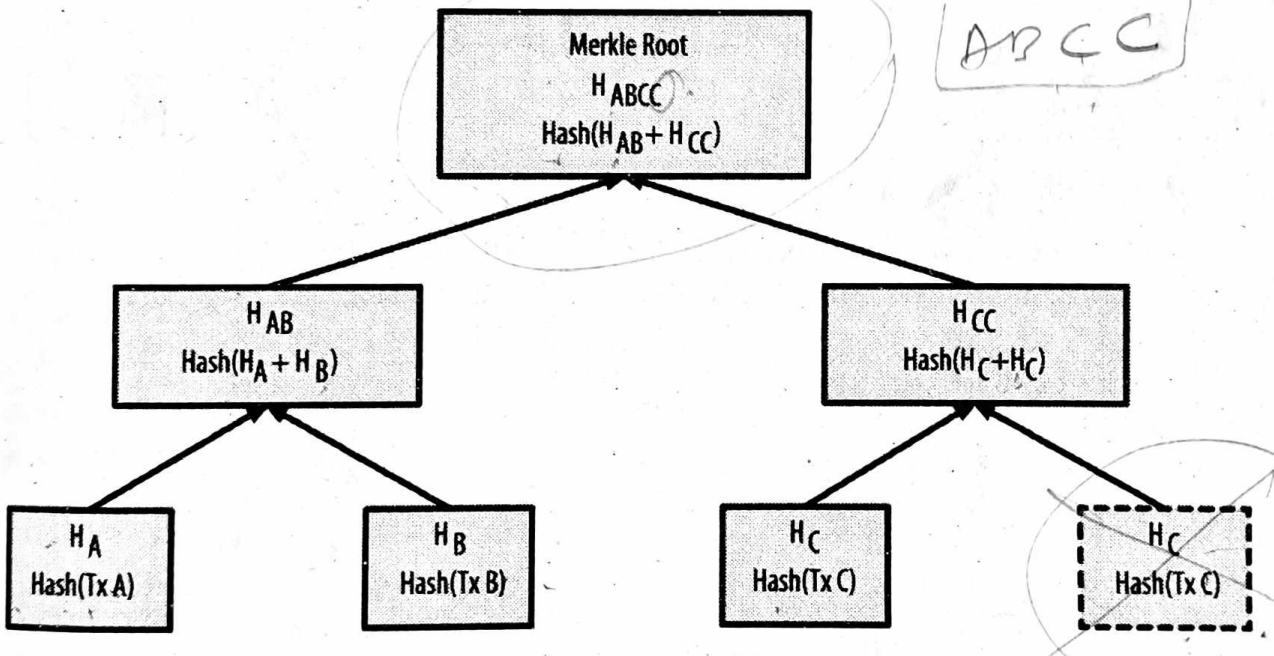


Fig. 1.11.3 : Duplicating one data element achieves an even number of data elements

- The same method for constructing a tree from four transactions can be generalized to construct trees of any size. In bitcoin it is common to have several hundred to more than a thousand transactions in a single block, which are summarized in exactly the

same way, producing just 32 bytes of data as the single Merkle root. In Fig. 1.11.4, you will see a tree built from 16 transactions. Note that although the root looks bigger than the leaf nodes in the diagram, it is the exact same size, just 32 bytes. Whether there is one transaction or a hundred thousand transactions in the block, the Merkle root always summarizes them into 32 bytes.

- To prove that a specific transaction is included in a block, a node only needs to produce $\log_2(N)$ 32-byte hashes, constituting an *authentication path* or *Merkle path* connecting the specific transaction to the root of the tree. This is especially important as the number of transactions increases, because the base-2 logarithm of the number of transactions increases much more slowly. This allows bitcoin nodes to efficiently produce paths of 10 or 12 hashes (320–384 bytes), which can provide proof of a single transaction out of more than a thousand transactions in a megabyte-size block.

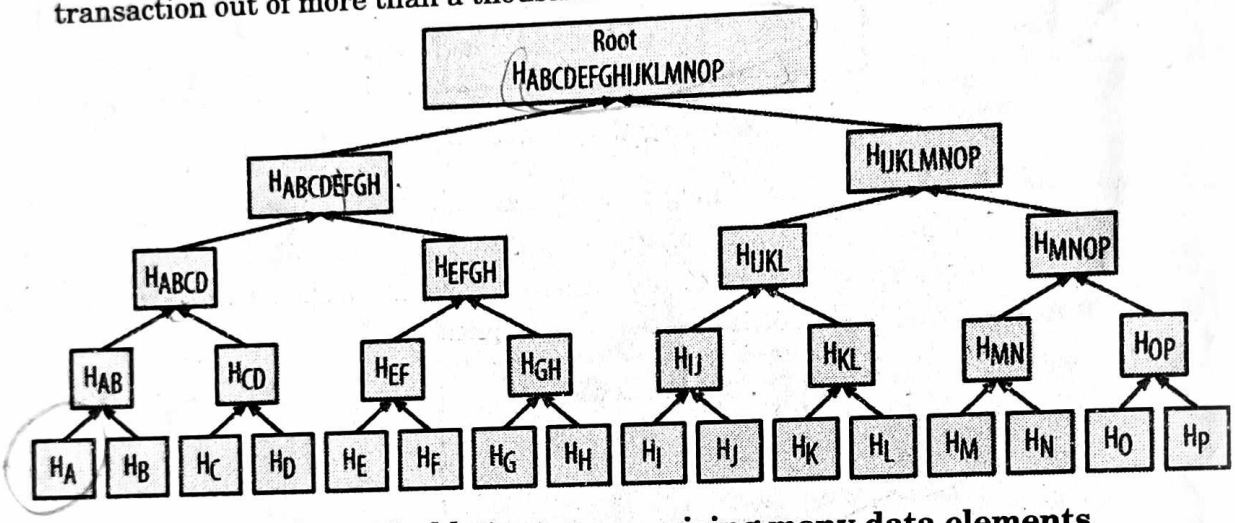


Fig. 1.11.4 : A Merkle tree summarizing many data elements

- In Fig. 1.11.5, a node can prove that a transaction K is included in the block by producing a Merkle path that is only four 32-byte hashes long (128 bytes total).

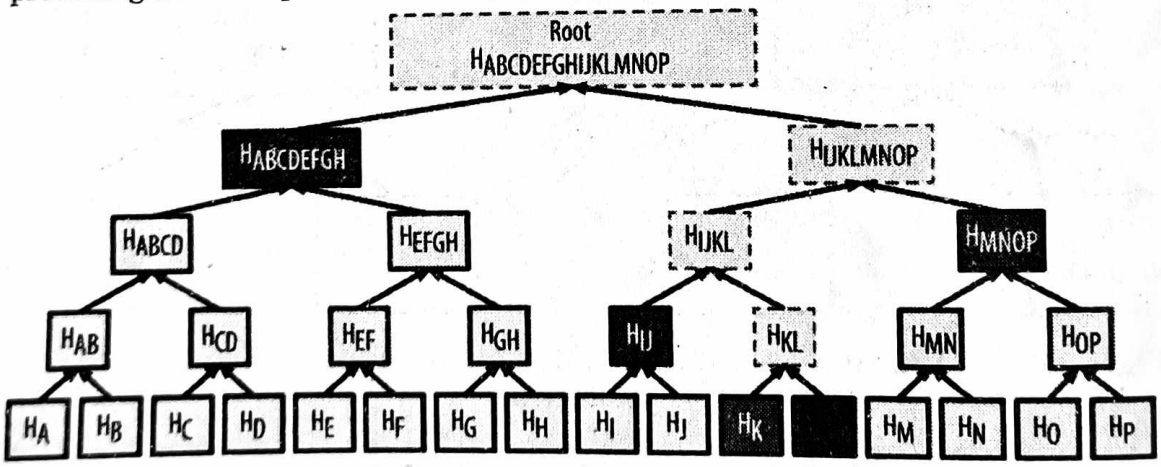


Fig. 1.11.5 : A Merkle path used to prove inclusion of a data element

- The path consists of the four hashes H_L, H_{IJ}, H_{MNOP} and $H_{ABCDEFGH}$. With those four hashes provided as an authentication path, any node can prove that H_K is included in the Merkle root by computing four additional pair-wise hashes H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, and the Merkle tree root.
- The code in Example 1-1 demonstrates the process of creating a Merkle tree from the leaf-node hashes up to the root, using the libbitcoin library for some helper functions.

Example 1-1 : Building a Merkle tree

```
#include <bitcoin/bitcoin.hpp>

bc::hash_digestcreate_merkle(bc::hash_list&merkle)
{
// Stop if hash list is empty.
if(merkle.empty())
returnbc::null_hash;
elseif(merkle.size()==1)
returnmerkle[0];

// While there is more than 1 hash in the list, keep looping...
while(merkle.size()>1)
{
// If number of hashes is odd, duplicate last hash in the list.
if(merkle.size()%2!=0)
merkle.push_back(merkle.back());
// List size is now even.
assert(merkle.size()%2==0);

// New hash list.
bc::hash_listnew_merkle;
// Loop through hashes 2 at a time.
for(autoit=merkle.begin();it!=merkle.end();it+=2)
{
// Join both current hashes together (concatenate).
bc::data_chunkconcat_data(bc::hash_size*2);
autoconcat=bc::make_serializer(concat_data.begin());
concat.write_hash(*it);
concat.write_hash(*(it+1));
```

```
assert(concat.iterator()==concat_data.end());
// Hash both of the hashes.
bc::hash_digestnew_root=bc::bitcoin_hash(concat_data);
// Add this to the new list.
new_merkle.push_back(new_root);
}
// This is the new list.
merkle=new_merkle;

// DEBUG output ----------------------------------
std::cout<<"Current merkle hash list:"<<std::endl;
for(constauto&hash:merkle)
std::cout<<" "<<bc::encode_hex(hash)<<std::endl;
std::cout<<std::endl;
// ----------------------------------------------
}
// Finally we end up with a single item.
returnmerkle[0];
}

intmain()
{
// Replace these hashes with ones from a block to reproduce the same merkle root.
bc::hash_listtx_hashes{{
bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000
00"),
bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000
11"),
bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000
22"),
}};
constbc::hash_digestmerkle_root=create_merkle(tx_hashes);
std::cout<<"Result: "<<bc::encode_hex(merkle_root)<<std::endl;
return0;
}
```

- Example 1-2 shows the result of compiling and running the Merkle code.

Example 1-2 : Compiling and running the Merkle example code

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the merkle executable
$ ./merkle
Current merklehash list:
 32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
 30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Current merklehash list:
 d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

- The efficiency of Merkle trees becomes obvious as the scale increases. Table 1.11.3 shows the amount of data that needs to be exchanged as a Merkle path to prove that a transaction is part of a block.

Table 1.11.3 : Merkle tree efficiency

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

- From the table, while the block size increases rapidly, from 4 KB with 16 transactions to a block size of 16 MB to fit 65,535 transactions, the Merkle path required to prove the inclusion of a transaction increases much more slowly, from 128 bytes to only 512 bytes.
- With Merkle trees, a node can download just the block headers (80 bytes per block) and still be able to identify a transaction's inclusion in a block by retrieving a small Merkle path from a full node, without storing or transmitting the vast majority of the blockchain, which might be several gigabytes in size.
- Nodes that do not maintain a full blockchain, called simplified payment verification (SPV nodes), use Merkle paths to verify transactions without downloading full blocks.

Chapter Ends...

❑❑❑