Module - 3

Consensus Algorithms and Bitcoin

***Consensus Algorithms*, Crash fault-tolerance (CFT) algorithms** – Paxos, Raft. **Byzantine fault-tolerance (BFT) algorithms** – Practical Byzantine Fault Tolerance (PBFT), Proof of work (PoW), Proof of stake (PoS), Types of PoS.

Bitcoin – Definition, Cryptographic keys – Private keys, public keys, addresses. **Transactions** – Lifecycle, Coinbase transactions, transaction validation. **Blockchain** – The genesis block. Mining – Tasks of miners, mining algorithm, hash rate. **Wallets** – Types of wallets.

Consensus

- Consensus is the backbone of a blockchain, as it provides the decentralization of control through an optional process known as **mining.**
- The choice of the **consensus algorithm** to utilize is governed by the type of blockchain in use; that is, not all consensus mechanisms are suitable for all types of blockchains.
- For example, in public permissionless blockchains, it would make sense to use PoW instead of mechanisms that are more suitable for permissioned blockchains, such as Proof of Authority (POA) or traditional Byzantine fault-tolerant consensus mechanisms.
- Therefore, it is essential to choose an appropriate consensus algorithm for a particular blockchain project.
- Consensus is a process of achieving agreement between distrusting nodes on the final state of data.
- To achieve consensus, different algorithms are used. It is easy to reach an agreement between two nodes (in client-server systems, for example), but when multiple nodes are participating in a distributed system and they need to agree on a single value, it becomes quite a challenge to achieve consensus.
- This process of attaining agreement on a common state or value among multiple nodes despite the failure of some nodes is known as **distributed consensus.**

Consensus mechanism

- A consensus mechanism is a set of steps that are taken by most or all nodes in a blockchain to agree on a proposed state or value.
- There are various requirements for a consensus mechanism. The following describes these requirements:
  - ✓ **Agreement:** All honest nodes decide on the same value.
  - ✓ **Integrity:** This is a requirement that no node can make the decision more than once in a single consensus cycle.
  - ✓ **Validity:** The value agreed upon by all honest nodes must be the same as the initial value proposed by at least one honest node.
  - ✓ **Fault tolerant:** The consensus algorithm should be able to run correctly in the presence of faulty or malicious nodes (Byzantine nodes).

- ✓ **Termination:** All honest nodes terminate the execution of the consensus process and eventually reach a decision.

Types of consensus mechanisms

- All consensus mechanisms are developed to deal with faults in a distributed system and to allow distributed systems to reach a final state of agreement.
- There are two general categories of consensus mechanisms. These categories deal with all types of faults (fail-stop types or arbitrary). These common types of consensus mechanisms are as follows:
  - ✓ **Proof-based consensus mechanisms:** This arrangement requires nodes to compete in a *leader-election lottery,* and the node that wins proposes the final value. The algorithm works on the principle of providing proof of some work and the possession of some authority or tokens to win the right of proposing the next block. For example, the PoW mechanism used in Bitcoin falls into this category, where a miner who solves the computational puzzle as proof of computational effort expended wins the right to add the next block to the blockchain.
  - ✓ **Traditional fault tolerance-based:** With no compute-intensive operations, such as *partial hash inversion* (as in Bitcoin PoW), this type of consensus mechanism relies on a simple scheme of nodes that publish and verify signed messages in a number of phases. Eventually, when a certain number of messages are received over a period of rounds (phases), then an agreement is reached.
- To achieve fault tolerance, replication is used. This is a standard and widely used method to achieve fault tolerance. In general, there are two types of faults that a node can experience:
  - ✓ **Fail-stop faults:** This type of fault occurs when a node merely has crashed. Fail-stop faults are the easier ones to deal with of the two fault types. Paxos or the RAFT protocol are normally used to deal with this type of fault. These faults are simpler to deal with.
  - ✓ **Byzantine faults:** The second type of fault is one where the faulty node exhibits malicious or inconsistent behavior arbitrarily. This type is difficult to handle since it can create confusion due to misleading information. This can be a result of an attack by adversaries, a software bug, or data corruption, SMR protocols such as ***Practical Byzantine Fault Tolerance (PBFT)*** was developed to address this second type of faults.
- Many other implementations of consensus protocols have been proposed in traditional distributed systems**. Paxos** is the most famous of these protocols. It was introduced by Leslie Lamport in 1989, With Paxos, nodes are assigned various roles such as Proposer, Acceptor, and Learner. Nodes or processes are named replicas, and consensus is achieved in the presence of faulty nodes by an agreement among a majority of nodes.
- An alternative to Paxos is RAFT, which works by assigning any of three states; that is. Follower Candidate, or Leader to the nodes. A Leader is elected after a Candidate node receives enough votes, and all changes then have to go through the Leader. The Leader commits the proposed changes once replication on the majority of the follower nodes is completed.

Fault tolerance

- A fundamental requirement in a consensus mechanism is that it must be fault-tolerant. In other words, it must be able to tolerate a number of failures in a network and should continue to work even in the presence of faults.
- This naturally means that there has to be some limit to the number of faults a network can handle, since no network can operate correctly if a large majority of its nodes are failing. Based on the requirement of fault tolerance, consensus algorithms are also called **fault-tolerant algorithms**, and there are two types of fault-tolerant algorithms.

Types of fault-tolerant consensus

- Fault-tolerant algorithms can be divided into two types of fault-tolerance. The first is **Crash fault-tolerance (CFT)** and the other is **Byzantine fault-tolerance (BFT)**.
- **CFT** covers only crash faults or, in other words, benign faults. In contrast, **BFT** deals with the type of faults that are arbitrary and can even be malicious.
- **Replication** is a standard approach to make a system fault-tolerant. Replication results in a synchronized copy of data across all nodes in a network.
- This technique improves the fault tolerance and availability of the network. This means that even if some of the nodes become faulty, the overall system/network remains available due to the data being available on multiple nodes.

There are two main types of replication techniques

- ✓ **Active replication,** which is a type where each replica becomes a copy of the original state machine replica.
- ✓ **Passive replication,** which is a type where there is only a single copy of the state machine in the system kept by the primary node, and the rest of the nodes/replicas only maintain the state.

Crash fault-tolerance (CFT) algorithms

- A distributed system faces a number of threats. Processes may crash, devices at some location may fail or a network connection may stop working. For enterprise use, the consensus algorithm must have resilience against a variety of threats.
- Crash fault tolerance (CFT) builds a degree of resiliency in the protocol, so that the algorithm can correctly take the process forward and reach consensus, even if certain components fail.
- CFT is a good solution when a component of the system fails. However, when blockchain is used by enterprises, there may be different companies, divisions or teams controlling separate components of the system. These different companies, divisions or teams may have different objectives, which leaves the system vulnerable to the threat of malicious activity. CFT is unable to reach consensus if any entity acts maliciously.

PAXOS

- Leslie Lamport developed Paxos. It is the most fundamental distributed consensus algorithm. allowing consensus over a value under unreliable communications. In other words, Paxos in used to build a reliable system that works correctly, even in the presence of faults.
- Paxos makes use of 2E+1 processes to ensure fault tolerance in a network where processes can crash fault, that is, experience *benign failures*. Benign failure means either the loss of a message or a process stops. In other words, Paxos can tolerate one crash failure in a three-node network
- Paxos is a two-phase protocol. The first phase is called the *prepare phase*, and the next phase is called the *accept phase*.
- Paxos has proposer and acceptors as participants, where the proposer the replicas or nodes that propose the values and acceptors are the nodes that accept the value.

How Paxos works

- The Paxos protocol assumes an asynchronous message-passing network with less than 50% of crash faults. As usual, the critical properties of the Paxos consensus algorithm are safety and liveness.
- Under safety, we have:
  - ✓ **Agreement**, which specifies that no two different values are agreed on. In other words, no two different learners learn different values.
  - ✓ **Validity**, which means that only the proposed values are decided. In other words, the values chosen or learned must have been proposed by a processor.
- Under liveness, we have
  - ✓ Termination, which means that, eventually, the protocol is able to decide and terminate. In other words, if a value has been chosen, then eventually learners will learn it.
- Processes can assume different roles, which are listed as follows:
  - ✓ **Proposers**, elected leader(s) that can propose a new value to be decided,
  - ✓ **Acceptors,** which participate in the protocol as a means to provide a majority decision.
  - ✓ **Learners,** which are nodes that just observe the decision process and value
- A single process in a Paxos network can assume all three roles
- The key idea behind Paxos is that the proposer node proposes a value, which is considered final only if a majority of the acceptor nodes accept it. The learner nodes also learn this final decision.
- Paxos can be seen as a protocol that is quite similar to the two-phase commit protocol. **Two- phase commit (2PC)** is a standard atomic commitment protocol to ensure that transactions are committed in distributed databases only if all participants agree to commit. Even if a single node cannot agree to commit the transaction, it is fully rolled back.
- Similarly, in Paxos, in the first phase, the proposer sends a proposal to the acceptors, if and when they accept the proposal, the proposer broadcasts a request to commit to the acceptors. Once the acceptors commit and report back to the proposer, the proposal is considered final, and the protocol concludes. In contrast with the two-phase commit, Paxos introduced ordering (sequencing to achieve total order) of the proposals and majority-based acceptance of the proposals instead of

expecting all nodes to agree (to allow progress even if some nodes fail). Both Of these improvements contribute toward ensuring the safety and liveness of the Paxos algorithm.

Paxos protocol works step by step:

1. The proposer proposes a value by broadcasting a message. **<*prepare(n)*>**, to all acceptors.
2. Acceptors respond with an acknowledgment message if proposal is the highest that the acceptor has responded to so far. The acknowledgment message **<*ack(n,v,s)*>** consists of three variables where ***n*** is the proposal number, ***v*** is the proposal value of the highest numbered proposal the acceptor has accepted so far, and ***s*** is the sequence number of the highest proposal accepted by the acceptor so far. This is where acceptors agree to commit the proposed value. The proposer now waits to receive acknowledgment messages from the majority of the acceptors indicating the **chosen value.**
3. If the majority is received, the proposer sends out the **"accept"** message **<accept(n, v)** to the acceptors.
4. If the majority of the acceptors accept the proposed value (now the "accept" message). then it is decided: that is, agreement is achieved.
5. Finally, in the learning phase, acceptors broadcast the **"accepted"** message **<*accepted(n,v)*>** to the proposer. This phase is necessary to disseminate which proposal has been finally accepted. The proposer then informs all other learners of the decided value. Alternatively, learners can learn the decided value via a message that contains the accepted value (decision value) multicast by acceptors.

- We can visualize this process in the following diagram

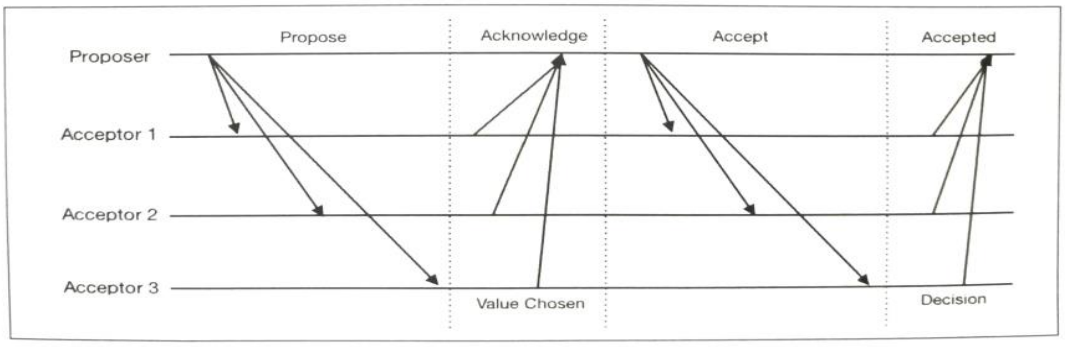


Figure : How Paxos works

How Paxos achieves safety and liveness

- ✓ **Agreement** is ensured by enforcing that only one proposal can win votes from a majority of the acceptors.
- ✓ **Validity** is ensured by enforcing that only the genuine proposals are decided. In other words, no value is committed unless it is proposed in the proposal message first.
- ✓ **Liveness or termination** is guaranteed by ensuring that at some point during the protocol execution, eventually there is a period during which there is only one fault-free proposer.

- First, a proposer suggests a value with the aim that acceptors achieve agreement on it.
- The decided value is a result of majority consensus among the acceptors and is finally learned by the learners.

RAFT

- The Raft protocol is a CFT consensus mechanism developed by Diego Ongaro and John Ousterhout at Stanford University, In Raft, the leader is always assumed to be honest.
- At a conceptual level, it is a replicated log for a **replicated state machine (RSM)** where a unique leader is elected every "term" (time division) whose log is replicated to all follower nodes.
- Raft is composed of three sub-problems
  - ✓ **Leader election** (a new leader election in case the existing one fails)
  - ✓ **Log replication** (leader to follower log synch)
  - ✓ **Safety** (no conflicting log entries (index) between servers)
- The Raft protocol ensures election safety, leader append only, log matching, leader completeness, and state machine safety.
- Each server in Raft can have either a **follower, leader, or candidate** state The protocol ensures election **safety** (that is, only one winner each election term) and **liveness**(that is, some candidate must eventually win).

How Raft works

The following steps will describe how the Raft protocol functions. At a fundamental level, the protocol is quite simple and can be described simply by the following sequence:

Node starts up-> Leader election > Log replication

1. First, the node starts up.
2. After this, the leader election process starts. Once a node is elected as leader, all changes go through that leader.
3. Each change is entered into the node's log.
4. Log entry remains uncommitted until the entry is replicated to follower nodes and the leader receives write confirmation votes from a majority of the nodes, then it is committed locally.
5. The leader notifies the followers regarding the committed entry.
6. Once this process ends, agreement is achieved.

The state transition of the Raft algorithm can be visualized in the following diagram:

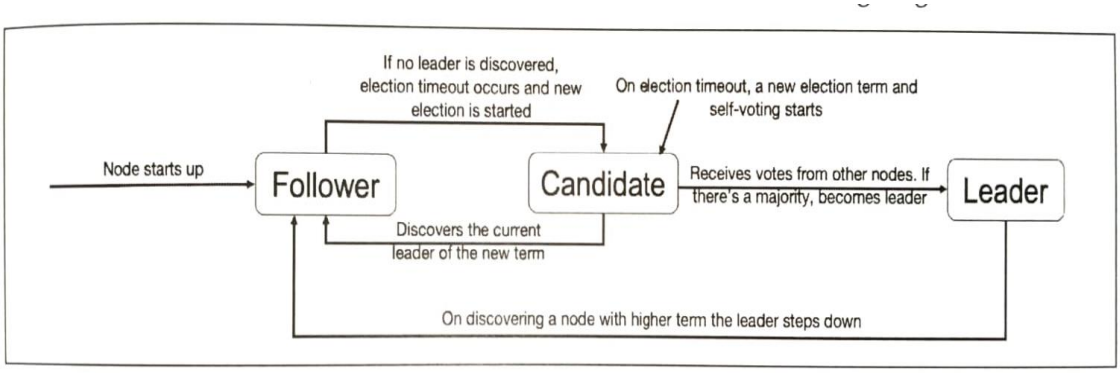


Figure: Raft state transition

The data is eventually replicated across all nodes in a consensus mechanism In Raft, the log (data) is eventually replicated across all nodes.

Log replication

Leg replication logic can be visualized in the following diagrams. The aim of log replication is to synchronize nodes with each other.

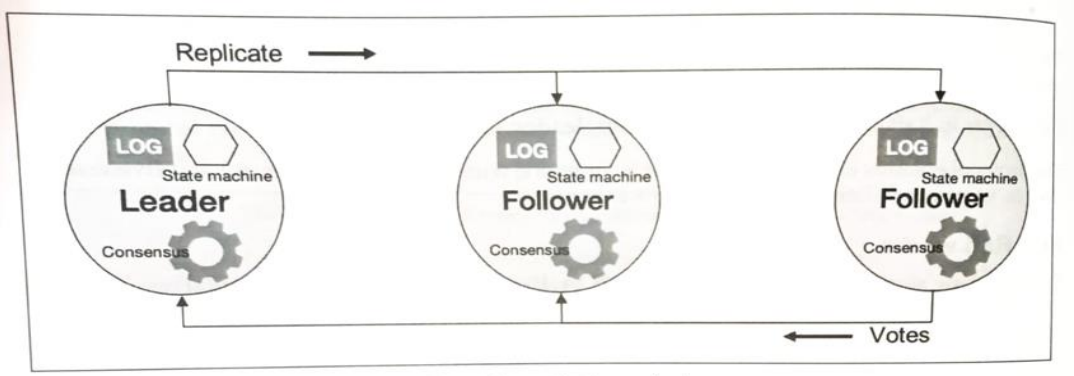


Figure : Log replication mechanism

- ✓ Log replication is a simple mechanism. As shown in the preceding diagram, the leader is responsible for log replication.
- ✓ Once the leader has a new entry in its log, it sends out the requests to replicate to the follower nodes When the leader receives enough confirmation votes back from the follower nodes indicating that the replicate request has been accepted and processed by the followers, the leader commits that entry to its local state machine.
- ✓ At this stage the entry is considered committed

Byzantine fault-tolerance (BFT) algorithms

Byzantine Fault Tolerance (BFT) is a consensus approach that resists a system to get into the Byzantine Generals' problem. It also means the system should stay intact even if one of the nodes (or general) fails. In addition, BFT aims to reduce the effect of malicious byzantine nodes (or general) on the network.

The consensus algorithm is how a blockchain achieves Byzantine Fault Tolerance. Since cryptocurrencies are decentralized, each one faces a large-scale version of the Byzantine Generals Problem. The blockchain needs to be able to function even if it has nodes that aren't working correctly or are providing false information.

Practical Byzantine Fault Tolerance (PBET)

- ➢ PBET was developed in 1999 by Miguel Castro and Barbara Liskov.
- ➢ PBFT, is a protocol developed to provide consensus in the presence of Byzantine faults, Before PBFT, Byzantine fault tolerance was considered impractical With PBFT, it was demonstrated for the first time that *practical Byzantine fault tolerance* is Possible.
- ➢ PBFT comprises three sub-protocols called **normal operation, view change**, and **checkpointing**.

- **Normal operation** sub-protocol refers to a scheme that is executed when everything is running normally and no errors are in the system **View change** is a sub-protocol that runs when a faulty leader node is detected in the system. **Checkpointing** is another sub-protocol, which in used to discard the old data from the system. The PBFT protocol comprises three phases or steps.
- These phases run in a sequence to achieve consensus. These phases are **pre-prepare, prepare, and commit,**
- The protocol runs in rounds where, in each round, an elected leader node, called the *primary node*, handles the communication with the client. In each round, the protocol plan through the three previously mentioned phases. The participants in the PBT protocol are of the replicas, where one of the replicas becomes primary as a leader in each round, and the rest of the nodes acts as backups,
- Here, each node maintains a local log, and the logs are kept in sync with each other via the consensus protocol that is, PBFT
- As we saw earlier, in order tolerate Byzantine faults, the minimum number of nodes required is **N-3F+1**, where **N** is the number of nodes and **F** is the number of faulty nodes PBFT ensures Byzantine fault tolerance as long as the number of nodes in a system stays **N-3F +1**
- We will now look at how the PBFT protocol works. In summary, when a client sends a request to a primary, the protocol initiates a sequence of operations between replicas, which eventually leads to consensus and a reply back to the client.
- These sequences of operations are divided into different phases:
    - ✓ Pre-prepare
    - ✓ Prepare
    - ✓ Commit

- In addition, each replica maintains a local state comprising three main elements
    - ✓ Service state
    - ✓ A message log
    - ✓ A number representing that replica's current view
- **<u>Pre-prepare:</u>**
- This is the first phase in the protocol, where the primary node, or primary, receives a request from the client. The primary node assigns a sequence number to the request. It then sends the pre-prepare message with the request to all backup replicas. When the pre-prepare message is received by the backup replicas, it checks a number of things to ensure the validity of the message:
    - ✓ First, whether the digital signature is valid.
    - ✓ After this, whether the current view number is valid.
    - ✓ Then, that the sequence number of the operation's request message is valid.
    - ✓ Finally, if the digest/hash of the operation's request message
- If all of these elements are valid, then the backup replica accepts the message. After accepting the message, it updates its local state and progresses toward the prepare phase.

- **Prepare:**
- Prepare message is sent by each backup to all other replicas in the system. Each backup waits for at least **2F+ I** prepare messages to be received from other replicas. They also check whether the prepare message contains the same view number, sequence number, and message digest values. If all these checks pass, then the replica updates its local state and progresses toward the commit phase.
- **Commit:**
- In the commit phase, each replica sends a commit message to all other replicas in the network. The same as the prepare phase, replicas wait for **2F + 1** commit messages to arrive from other replicas. The replicas also check the view number, sequence number, and message digest values. If they are valid for **2F+ 1** commit messages received from other replicas, then the replica executes the request, produces a result, and finally, updates its state to reflect a commit. If there are already some messages queued up, the replica will execute those requests first before processing the latest sequence numbers. Finally, the replica sends the result to the client in a reply message.
- The client accepts the result only after receiving 2F+ 1 reply messages containing the same result

How PBFT works

At a high level, the protocol works as follows:

1. A client sends a request to invoke a service operation in the primary,
2. The primary multicasts the request to the backups.
3. Replicas execute the request and send a reply to the client.
4. The client waits for replies from different replicas with the same result; this is the result of the operation.

The pre-prepare sub-protocol algorithm:

1. Accepts a request from the client.
2. Assigns the next sequence number.
3. Sends the pre-prepare message to all backup replicas.

The prepare sub-protocol algorithm

1. Accepts the pre-prepare message. If the backup has not accepted any pre-prepare messages for the same view or sequence number, then it accepts the message.
2. Sends the prepare message to all replicas.

Commit sub-protocol algorithm

1. The replica waits for **2F** prepare messages with the same view, sequence, and request
2. Sends a commit message to all replicas
3. Waits until a **2F+1** valid commit message arrives and is accepted.
4. Executes the received request.
5. Sends a reply containing the execution result to the client

- In summary, the primary purpose of these phases is to achieve consensus, where each phase is responsible for a critical part of the consensus mechanism, which after passing through all phases, eventually ends up achieving consensus
- The normal view of the PBFT protocol can be visualized as shown as follows:

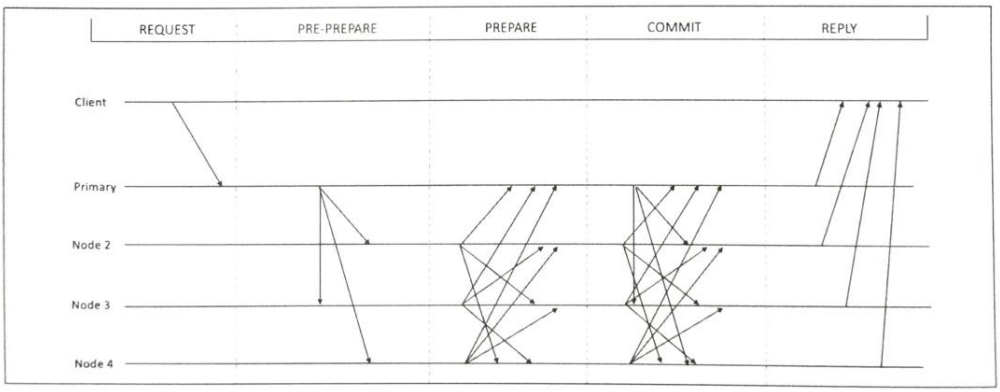


Figure : PBFT protocol

- During the execution of the protocol, the integrity of the messages and protocol operate be maintained to provide an adequate level of security and assurance. This is maintained by the use of digital signatures. In addition, certificates are used to ensure the adequate mat participants (nodes).

PBFT advantages and disadvantages:

Strengths:

- PBFT provides immediate and deterministic transaction finality. This is in contrast with the PoW protocol, where a number of confirmations are required to finalize a transaction with high probability.
- PBFT is also energy efficient as compared to PoW, which consumes a tremendous amount of electricity.

Weaknesses:

- PBFT is not very scalable. This is the reason it is more suitable for consortium networks, instead of public blockchains. It is, however, much faster than PoW protocols.
- Sybil attacks can be carried out on a PBFT network, where a single entity can control many identities to influence the voting and subsequently the decision. However, the fix is trivial and, in fact, this is not very practical in consortium networks where all identities are known on the network. This problem can be addressed simply by increasing the number of nodes in the network.

Proof of work (PoW) or Nakamoto consensus

- Nakamoto consensus, or PoW, was first introduced with Bitcoin in 2009. Since then, it has stood the test of time and is the longest-running blockchain network. This test of time in a testament to the efficacy of the PoW consensus mechanism.
- Now, we will explore how PoW works At a fundamental level, the PoW mechanism is designed to mitigate Sybil attacks, which facilitates consensus and the security of the network

- It is quite easy to obtain multiple identities and try to influence the network. However, in Bitcoin, due to the hashing power requirements, this attack is mitigated.

How PoW works

- In a nutshell, PoW works as follows:
  - PoW makes use of hash puzzles.
  - A node proposes a block has to find a nonce such that H (nonce || previous hash ||Tx|| Tx ||……..|| Tx) < Threshold value.

The process can be summarized as follows.

  - New transactions are broadcast to all nodes on the network
  - Each node collects the transactions into a candidate block
  - Miners propose new blocks
  - Miners concatenate and hash with the header of the previous block.
  - The resultant hash is checked against the target value, that is, the network difficulty target value.
  - If the resultant hash is less than the threshold value, then PoW is solved, otherwise, the nonce is incremented and the node tries again. This process continues until a resultant hash is found that is less than the threshold value.
- We can visualize how PoW works with the diagram shown here:

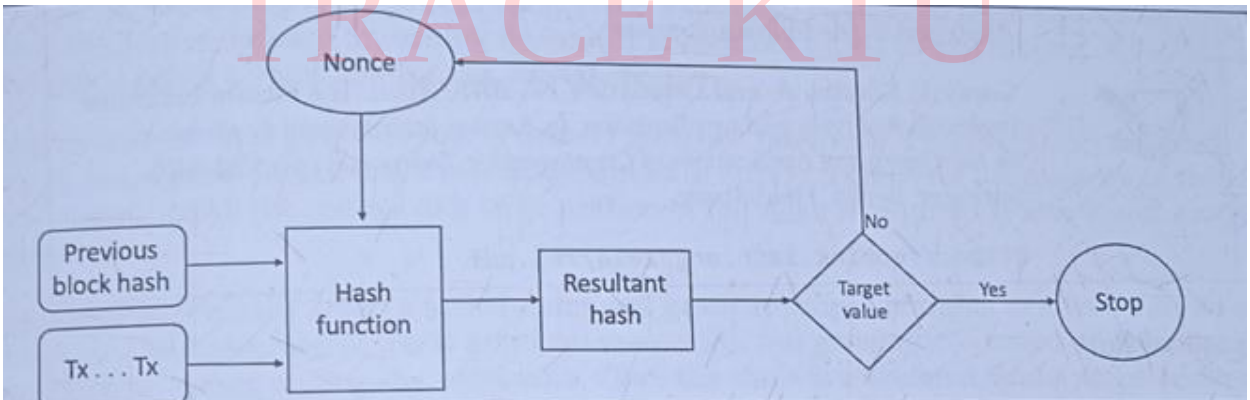


Figure 5.10: PoW diagram

PoW as a solution to the Byzantine generals problem

- The key idea behind PoW as a solution to the Byzantine generals problem is that all honest generals (miners in the Bitcoin world) achieve agreement on the same state (decision value).
- As long as honest participants control the majority of the network, PoW solves the Byzantine generals problem. Note that this is a probabilistic solution and not deterministic.

Nakamoto versus traditional consensus

- In traditional consensus algorithms, we have *agreement*, *validity*, and liveness properties which can be mapped to Nakamoto-specific properties of the *common prefix*, chain quality and **chain growth** properties respectively

- The common prefix property means that the blockchain hosted by honest modes will share the same large common prefix. If that is not the case, then the **agreement** property of the protocol cannot be guaranteed, meaning that the processors will not be able to decide and agree on the same value.
- The **chain quality** property means that the blockchain contains a certain required level of comm blocks created by honest nodes (miners) If chain quality is compromised, then the validity property of the protocol cannot be guaranteed. This means that there is a possibility that a value will be decided that is not proposed by a correct process, resulting in safely violation.
- The **chain growth** property simply means that new correct blocks are continuously added to th blockchain. If chain growth is impacted, then the liveness property of the protocol cannot be guaranteed. This means that the system can deadlock or fail to decide on a value.

Variants of PoW

There are two main variants of PoW algorithms, based on the type of hardware used for their processing.

- *CPU-bound PoW*
  - CPU-bound PoW refers to a type of PoW where the processing required to find the solution to the cryptographic hash puzzle is directly proportional to the calculation speed of the CPU or hardware such as ASICS Because ASICs have dominated the Bitcoin PoW and provide somewhat undue advantage to the miners who can afford to use ASICs, this CPU-bound Pow is seen as shifting toward centralization.
  - Moreover, mining pools with extraordinary hashing power can shift the balance of power towards them. Therefore, memory-bound PoW algorithms have been introduced, which are ASIC-resistant and are based on memory-oriented design instead of CPU
- *Memory-bound PoW*
  - Memory-bound PoW algorithms rely on system RAM to provide PoW. Here, the performance is bound by the access speed of the memory or the size of the memory. This reliance on memory also makes these PoW algorithms ASIC-resistant. Equihash is one of the most prominent memory-bound PoW algorithms.
  - Pow consumes tremendous amounts of energy; as such, there are several alternatives suggested by researchers. One of the first alternatives proposed in PoS.

Proof of Stake (PoS)

- The stake represents the number of coins (money) in the consensus protocol staked by a blockchain participant. The key idea is that if someone has a stake in the system, then they will not try to sabotage the system.
- Generally speaking, the chance of proposing the next block is directly proportional to the value staked by the participant. However, there are a few intricate variations. There are different variations of Po5, such as *chain-based PoS. committee-based PoS, BFT-based Pos, delegated PoS, leased PoS, and master node PoS.*

- In PoS systems, there is no concept of mining in the traditional Nakamoto consensus sense. However, the process related to earning revenue is sometimes called *virtual mining*. A POS miner is called either a *validator, minter, or stakeholder*. The right to win the next proposer role is usually assigned randomly. Proposers are rewarded either with transaction fees or block rewards.
- Similar to PoW, control over the majority of the network in the form of the control of a large portion of the stake is required to attack and control the network.
- PoS mechanisms generally select a stakeholder and grant appropriate rights to it based on their staked assets. The stake calculation is application-specific, but generally, is based on balance, deposit value, or voting among the validators. Once the stake is calculated, and a stakeholder is selected to propose a block, the block proposed by the proposer is accepted. The probability of selection increases with a higher stake. In other words, the higher the stake, the better the chances of winning the right to propose the next block.
- **How PoS work**

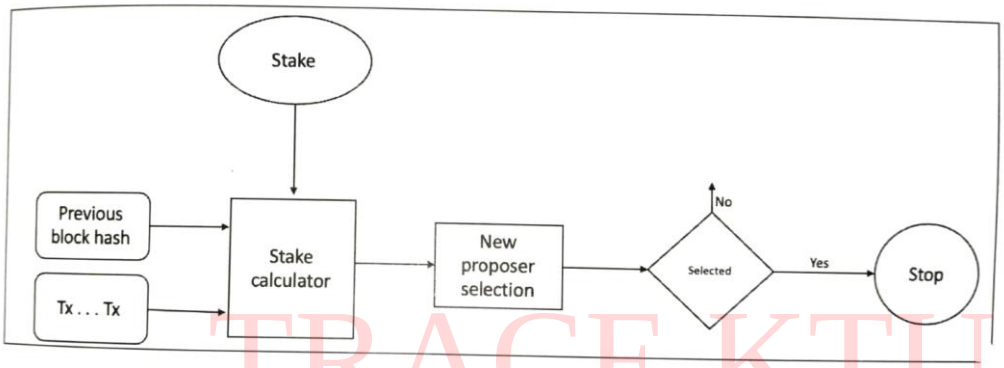


Figure : PoS diagram

- In the preceding diagram, a generic PoS mechanism is shown where a stake calculator function is used to calculate the amount of staked funds and, based on that, select a new proposer.

Types of PoS

There are several types of PoS algorithm.

Chain-based PoS

- This mechanism is very similar to Pow. The only change from the PoW mechanism is the block generation method. A block is generated in two steps by following a simple protocol:
  - ✓ Transactions are picked up from the memory pool and a candidate block is created.
  - ✓ A clock is set up with a constant tick interval, and at each clock tick, whether the hash of the block header concatenated with the clock time is less than the product of the target value and stake value is checked. This process can be shown in a simple formula:

Hash (B || clock time) < target x stake value

- The stake value is dependent on the way the algorithm is designed. In some systems, it is directly proportional to the amount of stake, and in others, it is based on the amount of time the stake has been held by the participant (also called coinage). The target is the mining difficulty per unit of the value of stake.

- This mechanism still uses hashing puzzles, as in PoW. But, instead of competing to solve the hashing puzzle by consuming a high amount of electricity and specialized hardware, the hashing puzzle in PoS is solved at regular intervals based on the clock tick. A hashing puzzle becomes easier to solve if the stake value of the minter is high.

➢ **Committee-based PoS**

- In this scheme, a group of stakeholders is chosen randomly, usually by using a **verifiable random function (VRF)**. This VRF, once invoked, produces a random set of stakeholders based on their stake and the current state of the blockchain.
- The chosen group of stakeholders becomes responsible for proposing blocks in sequential order. This mechanism is used in the Ouroboros PoS consensus mechanism, which is used in Cardano.

➢ **Delegated POS**

- DPoS is very similar to committee-based POS, but with one crucial difference. Instead of using a random function to derive the group of stakeholders, the group is chosen by stake delegation.
- The group selected is a fixed number of minters that create blocks in a round-robin fashion.
- Delegates are chosen via voting by network users. Votes are proportional to the amount of the stake that participants have on the network. This technique is used in Lisk, Cosmos, and EOS. DPoS is not decentralized as a small number of known users are made responsible for proposing and generating blocks.

Bitcoin

Definition

➢ Bitcoin can be defined in various ways; *it's a protocol, a digital currency, and a platform. It is a combination of a peer-to-peer network, protocols, and software that facilitates the creation and usage of the digital currency. Nodes in this peer-to-peer network talk to each other using the Bitcoin protocol.*

➢ Decentralization of currency was made possible for the first time with the invention of Bitcoin. Moreover, the double-spending problem was solved in an elegant and ingenious way in Bitcoin. The double-spending problem arises when, for example, a user sends coins to two different users at the same time and they are verified independently as valid transactions. The doublespending problem is resolved in Bitcoin by using a distributed ledger (the blockchain) where every transaction is recorded permanently, and by implementing a transaction validation and confirmation mechanism.

Cryptographic keys

On the Bitcoin network, possession of Bitcoins and the transfer of value via transactions are reliant upon private keys, public keys, and addresses. Elliptic Curve Cryptography (ECC) is used to generate public and private key pairs in the Bitcoin network.

Private keys in Bitcoin

- Private keys are required to be kept safe and normally reside only on the owner's side. Private keys are used to digitally sign the transactions, proving ownership of the bitcoins.
- Private keys are fundamentally 256-bit numbers randomly chosen in the range specified by the SECP256K1 ECDSA curve recommendation. Any randomly chosen 256-bit number from 0 x1 to 0x FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140 is a valid private key.
- Private keys are usually encoded using **Wallet Import Format (WIF)** in order to make them easier to copy and use. It is a way to represent the full-size private key in a different format. WIF can be converted into a private key and vice versa. The steps are described here.
  - For example, the consider the following private key:
    A3ED7EC8A03667180D01FB4251A546C2B9F2FE33507C68B7D9D4E1FA5714195201
  - When converted into WIF format, it looks as shown here:
    L2iN7umV7kbr6LuCmgM27rBnptGbDVc8g4ZBm6EbgTPQXnj1RCZP
  - ***mini private key format*** is sometimes used to create the private key with a maximum of 30 characters in order to allow storage where physical space is limited; for example, etching on physical coins or encoding in damage-resistant QR codes. The QR code is more damage resistant because more dots can be used for error correction and less for encoding the private key.
  - A private key encoded using mini private key format is also sometimes called a **minikey**. The first character of the mini private key is always the uppercase letter . A mini private key can be converted into a normal-sized private key, but an existing normal-sized private key cannot be converted into a mini private key. This format was used in **Casascius** physical bitcoins.
  - The Bitcoin core client also allows the encryption of the wallet that contains the private keys.
  - Public keys on their own are useless and private keys on their own are equally of no As we learned in Chapter 4, Public Key Cryptography,
  - Private keys have their own corresponding public keys. Public keys on their own are useless and private keys on their own are equally of no use. Pairs of public and private keys are required for the normal functioning of any public key cryptography-based systems such as the Bitcoin blockchain.

Public keys in Bitcoin

- Public keys exist on the blockchain and all network participants can see them. Public keys are derived from private keys due to their special mathematical relationship with those private keys.
- Once a transaction signed with the private key is broadcast on the Bitcoin network, public keys are used by the nodes to verify that the transaction has indeed been signed with the corresponding private key. This process of verification proves the ownership of the bitcoin.
- Bitcoin uses ECC based on the SECP256K1 standard. More specifically, it makes use of an **Elliptic Curve Digital Signature Algorithm (ECDSA)** to ensure that funds remain secure and can only be spent by the legitimate owner.
- Public keys can be represented in uncompressed or compressed format, and are fundamentally x and y coordinates on an elliptic curve. In uncompressed format, public keys are presented with a prefix of 0x4 in hexadecimal format.
- x and y coordinates are both 32 bytes in length. In total, a compressed public key is 33 bytes long, compared to the 65-byte uncompressed format. The compressed version of public keys include only the x part, since the y part can be derived from it.
- The reason why the compressed version of public keys works is that if the ECC graph is visualized, it reveals that the y coordinate can be either below the x axis or above the x axis, and as the curve is symmetric,
- only the location in the prime field is required to be stored. If y is even then its value lies above the x axis, and if it is odd then it is below the x axis. This means that instead of storing both x and y as the public key, only x needs to be stored with the information about whether y is even or odd.
- Initially, the Bitcoin client used uncompressed keys, but starting from Bitcoin Core client 0.6, compressed keys are used as standard. This resulted in an almost 50% reduction of space used to store public keys in the blockchain.
- Keys are identified by various prefixes, described as follows:
    - Uncompressed public keys use 0x04 as the prefix. Uncompressed public keys are 65 bytes long. They are encoded as 256-bit unsigned big-endian integers (32 bytes), which are concatenated together and finally prefixed with a byte 0x04. This means 1 byte for the 0x04 prefix, 32 bytes for the x integer, and 32 bytes for y integer, which makes it 65 bytes in total.
    - Compressed public keys start with 0x03 if the y 32-byte (256-bit) part of the public key is odd. It is 33 bytes in length as 1 byte is used by the 0x03 prefix (depicting an odd y) and Compressed public keys start with 0x02 if the y 32-byte (256-bit) part of the public key is even. It is 33 bytes in length as 1 byte is used by the ex02 prefix (depicting an even y) and 32 bytes for storing the x coordinate. 32 bytes for storing the x coordinate.
    - Compressed public keys start with 0x02 if the y 32-byte (256-bit) part of the public key is even. It is 33 bytes in length as 1 byte is used by the ex02 prefix (depicting an even y) and 32 bytes for storing the x coordinate.

Addresses in Bitcoin

- A Bitcoin address is created by taking the corresponding public key of a private key and hashing it twice, first with the SHA256 algorithm and then with RIPEMD160.
- The resultant 160-bit hash is then prefixed with a version number and finally encoded with a Base58Check encoding scheme. The Bitcoin addresses are 26-35 characters long and begin with digits 1 or 3.
- A typical Bitcoin address looks like the string shown here:
  1ANAguGG8bikEv2fYSTBnRUmx7QUcK58wt
- Addresses are also commonly encoded in a QR code for easy distribution.
- Currently, there are two types of addresses, the commonly used P2PKH and another P2SH type, starting with numbers 1 and 3, respectively.
- In the early days, Bitcoin used direct Pay-to-Pubkey, which is now superseded by P2PKH. However direct Pay-to-Pubkey is still used in Bitcoin for coinbase addresses.
- Addresses should not be used more than once; otherwise, privacy and security issues can arise.
- Avoiding address reuse circumvents anonymity issues to an extent, but Bitcoin has some other security issues as well, such as *transaction malleability, Sybil attacks, race attacks, and selfish mining,* all of which require different approaches to resolve.

❖ **Base58Check encoding**

- Bitcoin addresses are encoded using the Base58Check encoding. This encoding is used to limit the confusion between various characters, such as 0OII, as they can look the same in different fonts.
- The encoding basically takes the binary byte arrays and converts them into human readable strings. This string is composed by utilizing a set of 58 alphanumeric symbols.
- The following diagram shows how an address is generated, from generating the private key to the final output of the Bitcoin address:

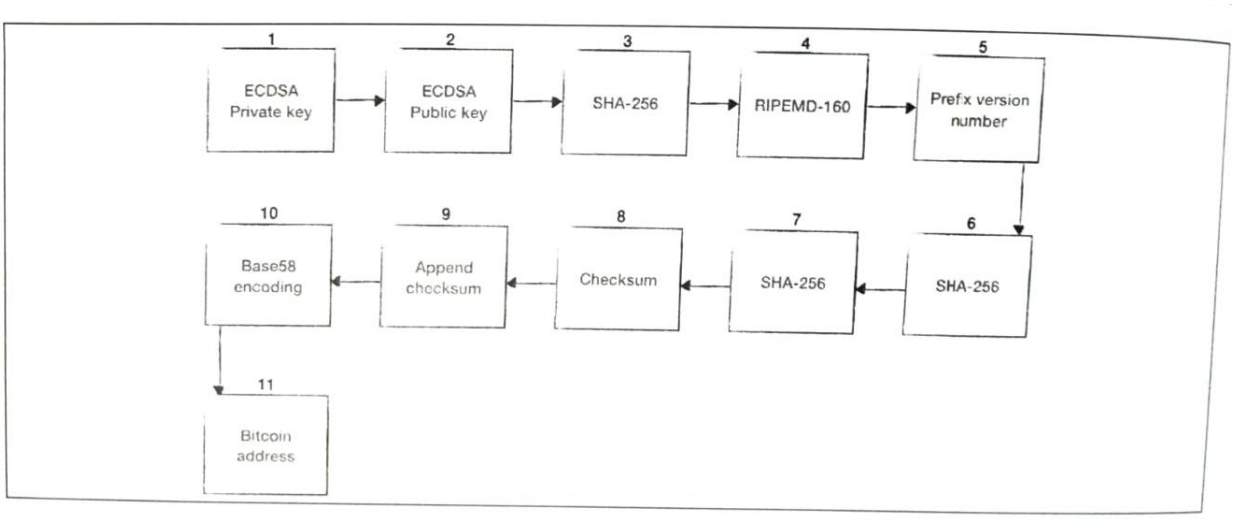


Figure : Address generation in Bitcoin

In the preceding diagram, there are several steps that we will now explain:

1. In the first step, we have a randomly generated ECDSA private key.

2. The public key is derived from the ECDSA private key.
3. The public key is hashed using the SHA-256 cryptographic hash function.
4. The hash generated in step 3 is hashed using the RIPEMD-160 hash function.
5. The version number is prefixed to the RIPEMD-160 hash generated in step 4.
6. The result produced in step 5 is hashed using the SHA-256 cryptographic hash function.
7. SHA-256 is applied again.
8. The first 4 bytes of the result produced from step 7 is the address checksum.
9. This checksum is appended to the RIPEMD-160 hash generated in step 4.
10. The resultant byte string is encoded into a Base58-encoded string by applying the Base58 encoding function.
11. Finally, the result is a typical Bitcoin address.

In addition to common types of addresses in Bitcoin, there are some advanced types of addresses available in Bitcoin too.

Vanity addresses

- As Bitcoin addresses are based on Base58 encoding, it is possible to generate addresses that contain human-readable messages. An example is shown as follows-note that in the first line, the name BasHir appears:
- Vanity addresses are generated using a purely brute-force method. An example of a paper wallet with a vanity address is shown in the following screenshot:

Figure : Vanity address generated from https://bitcoinvanitygen.com/

- In the preceding screenshot, on the right-hand bottom corner, the public vanity address is displayed with a QR code. The paper wallets can be stored physically as an alternative to the electronic storage of private keys.

Multi-signature addresses

- As the name implies, these addresses require multiple private keys. In practical terms, this means that in order to release the coins, a certain set number of signatures is required. This is also known as M of N **multisig.** Here, M represents the threshold or minimum number of signatures required from N number of keys to release the Bitcoins.

Bitcoin-Transactions

- Transactions are at the core of the Bitcoin ecosystem. Transactions can be as simple as just sending some bitcoins to a Bitcoin address, or can be quite complex, depending on the requirements.
- Each transaction is composed of at least one input and output. Inputs can be thought of as coins being spent that have been created in a previous transaction, and outputs as coins being created.
- If a transaction is minting new coins, then there is no input, and therefore no signature is needed. If a transaction should send coins to some other user (a Bitcoin address), then it needs to be signed by the sender with their private key.
- In this case, a reference is also required to the previous transaction to show the origin of the coins. Coins are unspent transaction outputs represented in Satoshis.
- Transactions are not encrypted and are publicly visible on the blockchain. Blocks are made up of transactions, and these can be viewed using any online blockchain explorer.

Transaction- Lifecycle

1. A user/sender sends a transaction using wallet software or some other interface.
2. The wallet software signs the transaction using the sender's private key.
3. The transaction is broadcasted to the Bitcoin network using a flooding algorithm.
4. Mining nodes (miners) who are listening for the transactions verify and include this transaction in the next block to be mined. Just before the transactions are placed in the block, they are placed in a special memory buffer called the transaction pool. The purpose of the transaction pool is explained in the next section.
5. Next, the mining starts, which is the process through which the blockchain is secured and new coins are generated as a reward for the miners who spend appropriate computational resources. Once a miner solves the PoW problem, it broadcasts the newly mined block to the network. PoW is explained in detail in the Mining section. The nodes verify the block and propagate the block further, and confirmations start to generate.
6. Finally, the confirmations start to appear in the receiver's wallet and after approximately three confirmations, the transaction is considered finalized and confirmed. However, three to six is just a recommended number; the transaction can be considered final even after the first confirmation. The key idea behind waiting for six confirmations is that the probability of double spending is virtually eliminated after six confirmations.

When a transaction is created by a user and sent to the network, it ends up in a special area on each Bitcoin software client. This special area is called the **transaction pool or memory pool**

➢ Transaction pool

- ✓ Also known as memory pools, these pools are basically created in local memory (computer RAM) by nodes (Bitcoin clients) in order to maintain a temporary list of transactions that have not yet been added to a block.
- ✓ Miners pick up transactions from these memory pools to create candidate blocks. Miners select transactions from the pool after they pass the verification and validity checks.
- ✓ The selection of which transactions to choose is based on the fee and their place in the order of transactions in the pool. Miners prefer to pick up transactions with higher fees.
- ✓ To send transactions on the Bitcoin network, the sender needs to pay a fee to the miners. This fee is an incentive mechanism for the miners.

➢ Transaction fees

- ✓ Transaction fees are charged by the miners. The fee charged is dependent upon the size and weight of the transaction. Transaction fees are calculated by subtracting the sum of the inputs from the sum of the outputs.
- ✓ A simple formula can be used:

  fee = sum(inputs) – sum(outputs)

➢ Transaction Data structure

- ✓ The transaction data structure at a high level contains metadata, inputs, and outputs. Transactions are combined create a block's body.

❖ **Metadata**

- This part of the transaction contains values such as the size of the transaction, the number of inputs and outputs, the hash of the transaction, and a lock_time field. Every transaction has a prefix specifying the version number. These fields are shown in the preceding example as lock_ time, size, and version.

❖ **Inputs**

- Generally, each input spends a previous output. Each output is considered an Unspent Transaction Output (UTXO) until an input consumes it. A UTXO is an unspent transaction output that can be spent as an input to a new transaction.

❖ **Outputs**

- Outputs have three fields, and they contain instructions for sending contains the amount of Satoshis, whereas the second field contains the size of the locking script. Finally, the third field contains a locking script that holds the conditions that need to be met in order for the output to be spent. More information on transaction spending using locking and producing outputs is discussed later in this section.

Coinbase transactions

- A coinbase transaction or generation transaction is always transaction created by a miner and is the first transaction in a block.
- It is used to create new coins. It includes a special field, also called the **coinbase**, which acts as an input to the coinbase transaction. This transaction also allows up to 100 bytes of arbitrary data storage.
- A coinbase transaction input has the same number of fields as a usual transaction input, but the structure contains the coinbase data size and fields instead of the unlocking script size and fields.
- Also, it does not have a reference pointer to the previous transaction. This structure is the following table:

Field	Size	Description
Transaction hash	32 bytes	Set to all zeroes as no hash reference is used
Output index	4 bytes	Set to `0xFFFFFFFF`
Coinbase data length	1-9 bytes	2-100 bytes
Data	Variable	Any data
Sequence number	4 bytes	Set to `0xFFFFFFFF`

- All transaction in the bitcoin system go under a validation mechanism.

Transaction validation

- This verification process is performed by Bitcoin nodes. There are three main things that nodes check when verifying a transaction:
  1. That transaction inputs are previously unspent. This validation step prevents double spending by verifying that the transaction inputs have not already been spent by someone else.
  2. That the sum of the transaction outputs is not more than the total sum of the transaction inputs. However, both input and output sums can be the same, or the sum of the input (total value) could be more than the total value of the outputs. This check ensures that no new bitcoins are created out of thin air.
  3. That the digital signatures are valid, which ensures that the script is valid.

Even though transaction construction and validation are generally a secure and sound process, some vulnerabilities exist in Bitcoin.

Transaction bugs

The following are two major Bitcoin vulnerabilities that have been infamously exploited.

- **Transaction malleability**
  - ✓ Transaction malleability is a Bitcoin attack that was introduced due to a bug in the Bitcoin implementation. Due to this bug, it became possible for an adversary to change the transaction ID of a transaction, thus resulting in a scenario where it would appear that a certain transaction has not been executed.
  - ✓ This can allow scenarios where double deposits or withdrawals can occur. In other words, this bug allows the changing of the unique ID of a Bitcoin transaction before it is confirmed. If the ID is changed before confirmation without making the transaction invalid, it would seem that the transaction did not occur at all, which can then give the false impression that the transaction has not been executed, thus allowing double-deposit or withdrawal attacks.
- **Value overflow**
  - ✓ This incident is one of the most well-known events in Bitcoin history. On 15 August 2010, a transaction was discovered that created roughly 184 billion bitcoins. This problem occurred due to the integer overflow bug where the amount field in the Bitcoin code was defined as a signed integer instead of an unsigned integer.
  - ✓ This bug means that the amount can also be negative, and resulted in a situation where the outputs were so large that the total value resulted in an overflow.
  - ✓ To the validation logic in Bitcoin code, all appeared to be correct, and it looked like the fee was also positive (after the overflow). T
  - ✓ his bug was fixed via a soft fork (more on this in overflow. Blockchain section) quickly after its discovery.

Blockchain

The blockchain can be defined as a public, distributed ledger holding a timestamped, ordered, and immutable record of all transactions on the Bitcoin network. Transactions are picked up by miners and bundled into blocks for mining. Each block is identified by a hash and is linked to its previous block by referencing the previous block's hash in its header.

- The data structure of a bitcoin block is shown following table.

Field	Size	Description
Block size	4 bytes	The size of the block.
Block header	80 bytes	This includes fields from the block header described in the next section.
Transaction counter	Variable	The field contains the total number of transactions in the block, including the coinbase transaction. Size ranges from 1-9 bytes.
Transactions	Variable	All transactions in the block.

- The block header mentioned in the previous table is a data structure that contains several fields. This is shown in the follow ing table:

Field	Size	Description
Version	4 bytes	The block version number that dictates the block validation rules to follow.
Previous block's header hash	32 bytes	This is a double SHA256 hash of the previous block's header.
Merkle root hash	32 bytes	This is a double SHA256 hash of the Merkle tree of all transactions included in the block.
Timestamp	4 bytes	This field contains the approximate creation time of the block in Unix-epoch time format. More precisely, this is the time when the miner started hashing the header (the time from the miner's location).
Difficulty target	4 bytes	This is the current difficulty target of the network/block.
Nonce	4 bytes	This is an arbitrary number that miners change repeatedly to produce a hash that is lower than the difficulty target.

- The following diagram provides a detailed view of the blockchain structure. As shown in the following diagram, 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:

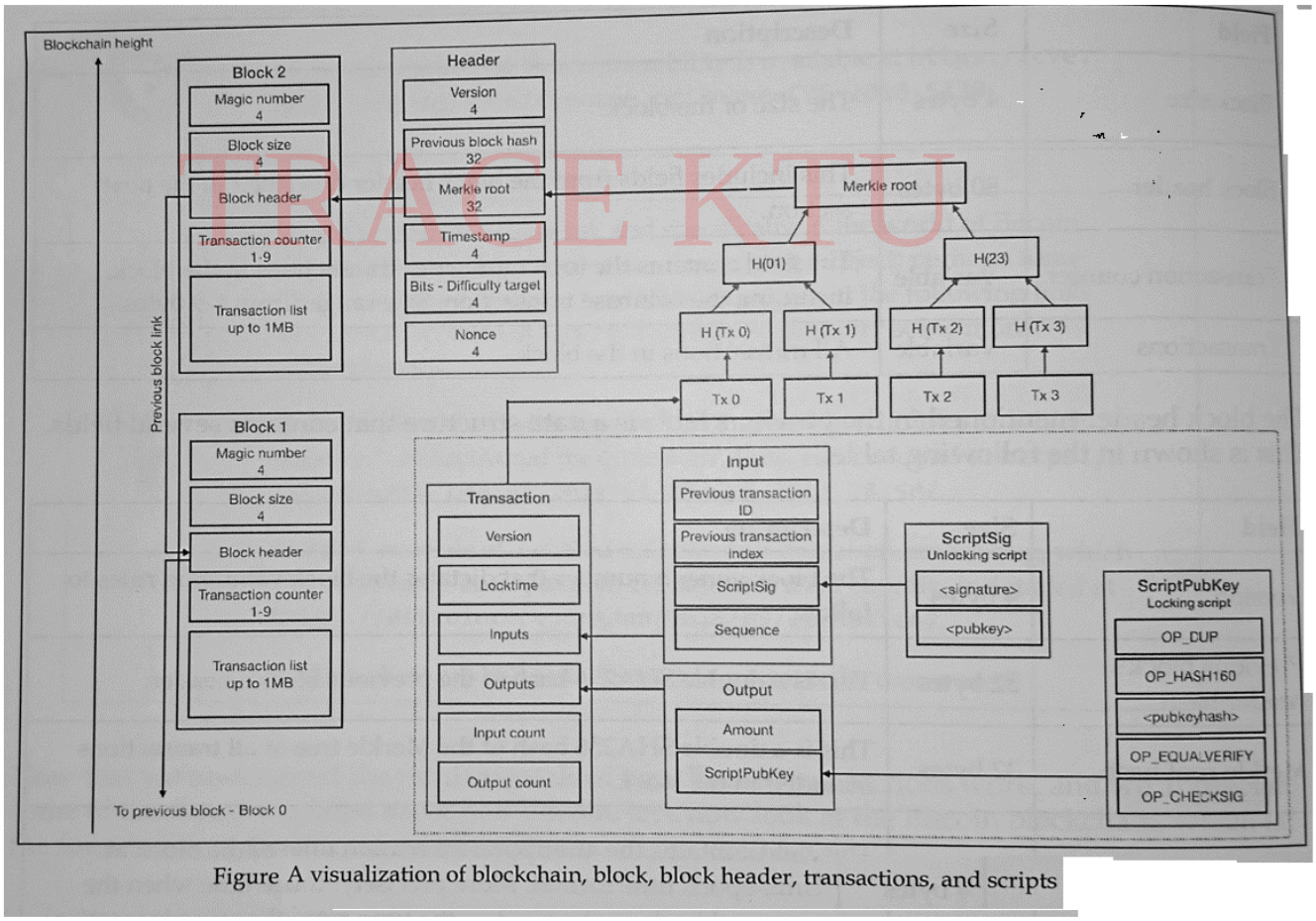


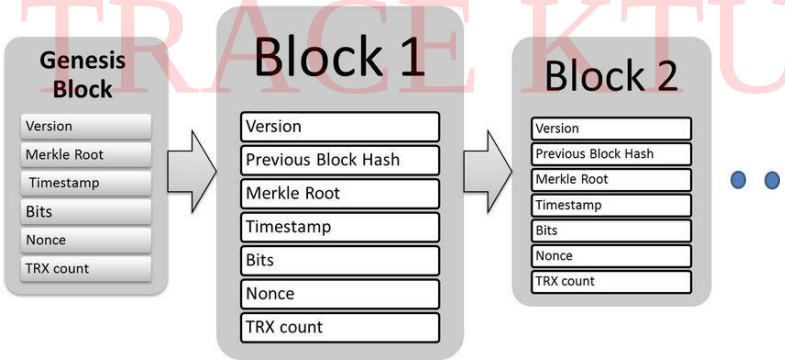
Figure A visualization of blockchain, block, block header, transactions, and scripts

- The preceding diagram shows a high-level overview of the Bitcoin blockchain. On the left-hand side, blocks are shown starting from bottom to top. Each block contains transactions and block headers, which are further magnified on the right-hand side.
- At the top, first, the block header Merkle root element of the block header is shown in magnified view, which shows how Merkle is enlarged to show various elements within the block header.

- Then on the right-hand side, the root is constructed. Further down the diagram, transactions are also magnified to show the structure of a transaction and the elements that it contains.
- Also, note that transactions are then further elaborated to show what locking and unlocking scripts look like. The size (in bytes) of each field of block, header and transaction is also shown as a number under the name of the field.

Genesis Block

- A Genesis Block is the name given to the first block a cryptocurrency, such as Bitcoin, ever mined. A blockchain consists of a series of blocks that are used to store information related to transactions that occur on a blockchain network. Each of the blocks contains a unique header, and each such block is identified by its block header hash individually.
- These blocks get layered—one on top of the other, with the Genesis Block being the foundation—and they grow in height until the end of the blockchain is reached and the sequence is complete. The layers and deep history of each sequence is one of the things that makes a blockchain-based cryptocurrency so secure.
- Bitcoin's Genesis Block was the first instance of a proof-of-work blockchain system and is the template for all other blocks in its blockchain. In 2009, Bitcoin's pseudonymous developer, Satoshi Nakamoto, created the Genesis Block, which launched the cryptocurrency boom that is ongoing today.



- The Genesis Block, also known as Block 0, is the very first block upon which additional blocks in a blockchain are added. It is effectively the ancestor that every other block can trace its lineage back to since every block references the one preceding it. This began the process of validating bitcoin transactions and introducing new bitcoins into existence.

Mining

- Mining is a process by which new blocks are added to the blockchain. Blocks contain transactions that are validated via the mining process by mining nodes on the Bitcoin network. Blocks, once mined and verified, are added to the blockchain, which keeps the blockchain growing.
- This process is resource-intensive due to the requirements of PoW, where miners compete to find a number less than the difficulty target of the network. This difficulty in finding the correct value

(also called sometimes the mathematical puzzle) is there to ensure that miners have spent the required resources before a new proposed block can be accepted.

- The miners mint new coins by solving the PoW problem, also known as the partial hash inversion problem. This process consumes a high amount of resources, including computing power and electricity. This process also secures the system against fraud and double-spending attacks while adding more virtual currency to the Bitcoin ecosystem.
- Roughly one new block is created (mined) every 10 minutes to control the frequency of generation of bitcoins. This frequency needs to be maintained by the Bitcoin network. It is encoded in the Bitcoin Core client to control the "money supply."

Tasks of the miners

Once a node connects to the Bitcoin network, there are several tasks that a Bitcoin miner performs:

1. **Synching up with the network:** Once a new node joins the Bitcoin network, it downloads the blockchain by requesting historical blocks from other nodes. This is mentioned here in the context of the Bitcoin miner; however, this not necessarily a task that only concerns miners.
2. **Transaction validation:** Transactions broadcast on the network are validated by full nodes by verifying and validating signatures and outputs.
3. **Block validation:** Miners and full nodes can start validating blocks received by them by evaluating them against certain rules. This includes the verification of each transaction in the block along with verification of the nonce value.
4. **Perform Pow:** This task is the core of the mining process and this is where miners find a valid block by solving a computational puzzle. The block header contains a 32-bit nonce field and miners are required to repeatedly vary the nonce until the resultant hash is less than a predetermined target.
5. **Fetch reward:** Once a node solves the hash puzzle (PoW), it immediately broadcasts the results, and other nodes verify it and accept the block. There is a slight chance that the newly minted block will not be accepted by other miners on the network due to a clash with another block found at roughly the same time, but once accepted, the miner is rewarded with 12.5 bitcoins and any associated transaction fees.

Mining algorithm

The mining algorithm consists of the following steps:

1. The previous block's header is retrieved from the Bitcoin network.
2. Assemble a set of transactions broadcast on the network a block to be proposed.
3. Compute the double hash of the previous block's header, combined with a nonce and the newly proposed block, using the SHA256 algorithm.
4. Check if the resulting hash is lower than the current difficulty level (the target), then PoW is solved. As a result of successful PoW, the discovered block is broadcasted to the network and miners fetch the reward.

5. If the resultant hash is not less than the current difficulty level (target), then repeat the process after incrementing the nonce.

- As the hash rate of the Bitcoin network increased, the total amount of the 32-bit nonce was exhausted too quickly. In order to address this issue, the extra nonce solution was implemented, whereby the coinbase transaction is used to provide a larger range of nonce to be searched by the miners.

This process is visualized in the following flowchart:

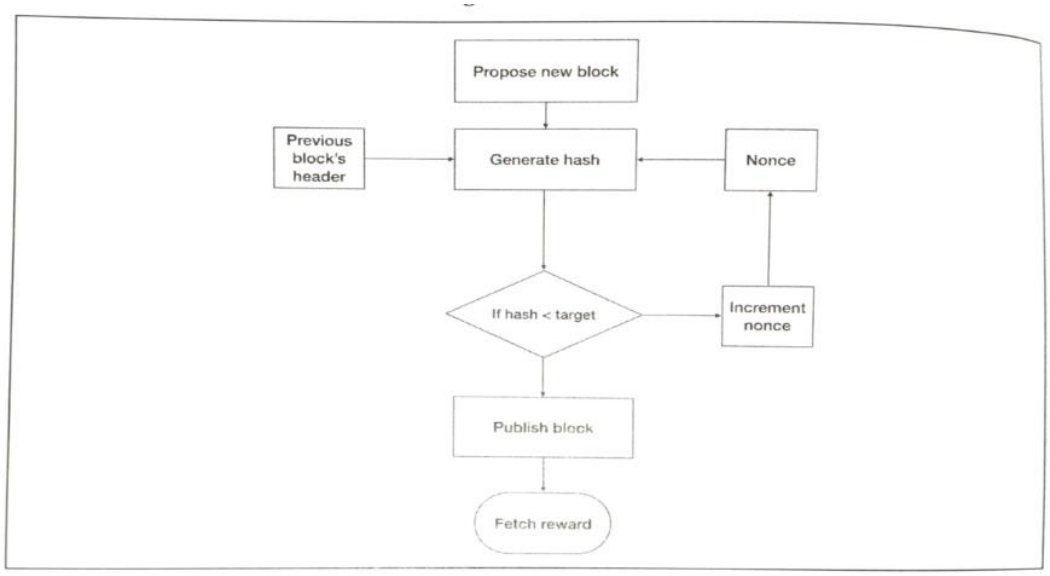


Figure :Mining process

- Mining difficulty increases over time and bitcoins that could once be mined by a single-CPU laptop computer now require dedicated mining centers to solve the hash puzzle.

Hash Rate

- The hashrate is an important metric for assessing the strength of a blockchain network – more specifically, its security. The more machines dedicated by honest miners to discovering the next block, the higher the hashrate rises and the harder it becomes for malicious agents to disrupt the network.
- A 51% attack, for example, is when a single individual or group of attackers purchases or rents enough mining equipment to control over 50% of a blockchain's hashrate. Because blockchains are trustless and abide by a rule known as the "longest chain rule," a person or group that controls a majority of the hashrate could, in theory, block or reorganize transactions and even reverse their own payments.
- This would create double spend issues which, in turn, would completely undermine the integrity of the underlying blockchain. A fall in hashrate, therefore, means a reduction in the cost to perform a 51% attack, making the network more vulnerable.

Wallets

- The wallet software is used to generate and store cryptographic keys. It performs various useful functions, such as receiving and sending Bitcoin, backing up keys, and keeping track of the balance available. Bitcoin client software usually offers both functionalities: Bitcoin client and wallet.
- Private keys are generated by randomly choosing a 256-bit number provided by the wallet software. Private keys are used by wallets to sign the outgoing transactions.
- Wallets do not store any coins, and there is no concept of wallets storing balance or coins for a user. In fact, in the Bitcoin network, coins do not exist; instead, only transaction information is stored on the blockchain (more precisely, UTXO, unspent outputs), which are then used to calculate the number of bitcoins.
- In Bitcoin, there are different types of wallets that can be used to store private keys. As software, they also provide some functions to the users to manage and carry out transactions on the Bitcoin network.

Types of Wallets

1. **Non-deterministic wallets**
   - These wallets contain randomly generated *private keys* and are also called Just a *bunch of Key wallets.* The Bitcoin Core client generates some keys when first started and also generates keys as and when required.
   - Managing a large number of keys is very difficult and an error-prone process that can lead to the theft and loss of coins. Moreover, there is a need to create regular backups of the keys and protect them appropriately, for example, by encrypting them in order to prevent theft or loss.
2. **Deterministic wallets**
   - In this type of wallet, keys are derived from a *seed value via hash functions*. This seed number is generated randomly and is commonly represented by human-readable mnemonic code words.
3. **Hardware wallets**
   - Another method is to use a tamper-resistant device to store keys. This tamper-resistant device can be custom-built. With the advent of NFC-enabled phones, this can also be a secure element (SE) in NFC phones. Trezor and Ledger wallets (various types) are the most commonly used Bitcoin hardware wallets:
4. **Online wallets**
   - Online wallets, as the name implies, are stored entirely online and are provided as a service usually via the cloud. They provide a web interface to the users to manage their wallets and perform various functions, such as making and receiving payments. They are easy to use but imply that the user trusts the online wallet service provide.
5. **Mobile wallets**
   - Mobile wallets, as the name suggests, are installed on mobile devices. They can provide us with various methods to make payments, most notably the ability to use smartphone cameras to scan QR codes quickly and make payments.
   - Mobile wallets are available for Android and iOS and include Blockchain Wallet, Breadwallet, Copay, and Jaxx:

Important questions

1. Explain how proof of stake can achieve consensus among peers?
2. If your blockchain network has 5 byzantine nodes, what is the minimum number of nodes that are required to ensure Byzantine fault tolerance using PBFT protocol?
3. How are transactions verified in a Bitcoin network?
4. Explain and illustrate how Paxos protocol can be used to achieve consensus.
5. Show how practical Byzantine Fault Tolerance can achieve consensus in the presence of Byzantine faults.
6. Describe the various fields that make up a transaction in Bitcoin.
7. What is the role of a Bitcoin miner? Explain the mining algorithm used in Bitcoin with the help of a flowchart?