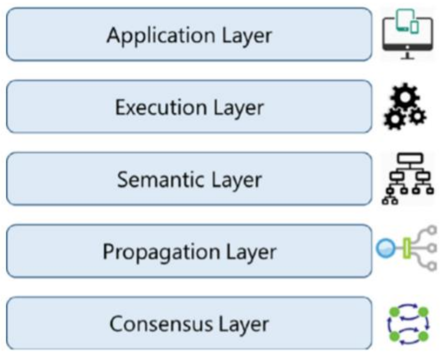
1) Define Blockchain, explain layers and structure of Blockchain.

ans) Blockchain is a system of records to transact value (not just money!) in

a peer-to-peer fashion.



Various layers of blockchain

Application Layer

This is the layer where you code up the desired functionalities and make an application out of it for the end users. It usually involves a traditional tech stack for software development such as client-side programming constructs, scripting, APIs, development frameworks, etc. For the applications that treat blockchain as a backend, those applications might need to be hosted on some web servers and that might require web application development, server-side programming, and APIs, etc. Ideally, good blockchain applications do not have a client–server model, and there are no centralized servers that the clients access, which is just the way Bitcoin works

Execution Layer

The Execution Layer is where the executions of instructions ordered by the Application Layer take place on all the nodes in a blockchain network. The instructions could be simple instructions or a set of multiple instructions in the form of a smart contract. In either case, a program or a script needs to be executed to ensure the correct execution of the transaction. All the nodes in a blockchain network have to execute the programs/scripts independently. Deterministic execution of programs/scripts on the same set of inputs and conditions always produces the same output on all the nodes, which helps avoid inconsistencies.

Semantic Layer

The Semantic Layer is a logical layer because there is an orderliness in the transactions and blocks. A transaction, whether valid or invalid, has a set of instructions that gets through the Execution Layer but gets validated in the Semantic Layer. If it is Bitcoin, then whether one is spending a legitimate transaction, whether it is a double-spend attack, whether one is authorized to make this transaction, etc., are validated in this layer.

Propagation Layer The previous layers were more of an individual phenomenon: not much coordination with other nodes in the system. The Propagation Layer is the peer-to-peer communication layer that allows the nodes to discover each other, and talk and sync with each other with respect to the current state of Chapter 1 Introduction to Blockchain 22 the network. When a transaction is made, we know that it gets broadcast to the entire network. Similarly, when a node wants to propose a valid block, it gets immediately propagated to the entire network so that other nodes could build on it, considering it as the latest block. So, transaction/block propagation in the network is defined in this layer, which ensures stability of the whole network. By design, most of the blockchains are designed such that they forward a transaction/block immediately to all the nodes they are directly connected to, when they get to know of a new transaction/block

Consensus Layer

The Consensus Layer is usually the base layer for most of the blockchain systems. The primary purpose of this layer is to get all the nodes to agree on one consistent state of the ledger. There could be different ways of achieving consensus among the nodes, depending on the use case. Safety and security of the blockchain is accertained in this layer.

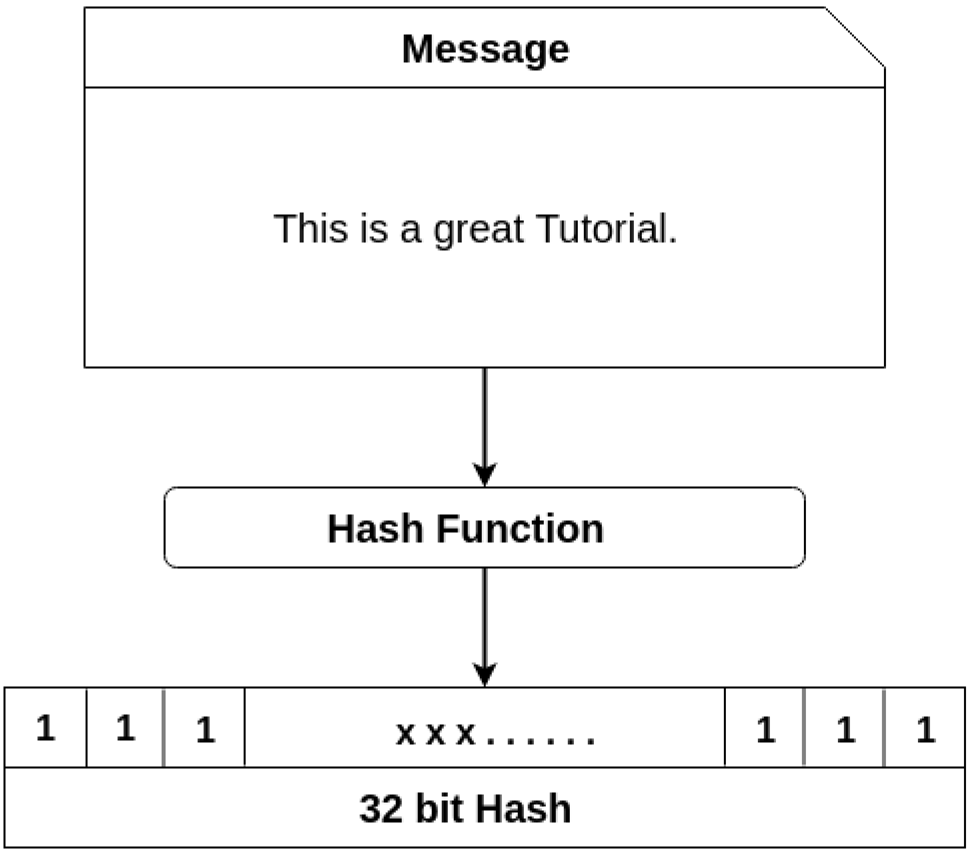
2) Explain cryptography. Explain the types of cryptography.
   Ans)
   Cryptography is the science of protecting information by transforming it into a secure format.
   there are two kinds of cryptography: symmetric key and asymmetric key (a.k.a. public key) cryptography
   Symmetric Key Cryptography
   In the previous section we looked at how Alice can encrypt a message and send the ciphertext to Bob. Bob can then decrypt the ciphertext to get the original message. If the same key is used for both encryption and decryption, it is called symmetric key cryptography.
   This means that both Alice and Bob have to agree on a key (k) called "shared secret" before they exchange the ciphertext. So, the process is as follows:
   Alice—the Sender:
   • Encrypt the plaintext message m using encryption algorithm E and key k to prepare the ciphertext c
   • c = E(k, m)
   • Send the ciphertext c to Bob
   Bob—the Receiver:
   • Decrypt the ciphertext c using decryption algorithm D and the same key k to get the plaintext m
   • m = D(k, c)
   Did you just notice that the sender and receiver used the same key (k)? How do they agree on the same key and share it with each other? Obviously, they need a secure distribution channel to share the key.
   Symmetric key cryptography is used widely; the most common uses are secure file transfer protocols such as HTTPS, SFTP, and Web DAVS. Symmetric cryptosystems are usually faster and more useful when the data size is huge.
   Asymmetric Key Cryptography
   Asymmetric key cryptography, also known as "public key cryptography," is a revolutionary concept introduced by Diffie and Hellman. With this technique, they solved the problem of key distribution in a symmetric cryptography system by introducing digital signatures. Note that asymmetric key cryptography does not eliminate the need for symmetric key cryptography.
   Assume that Alice wants to send a message to Bob confidentially so that no one other than Bob can make sense of the message, then it would require the following steps:
   Alice—The Sender:

• Encrypt the plaintext message m using encryption algorithm E and the public key PukBob to prepare the ciphertext c.
• c = E(PukBob, m)
• Send the ciphertext c to Bob.
Bob—The Receiver:
• Decrypt the ciphertext c using decryption algorithm D and its private key PrkBob to get the original plaintext m.
• m = D(PrkBob, c)
Notice that the public key should be kept in a public repository accessible to everyone and the private key should be kept as a well guarded secret. Public key cryptography also provides a way of authentication

3) Explain hash functions with code examples.

Blockchain Hash Function

A hash function takes an input string (numbers, alphabets, media files) of any length and transforms it into a fixed length. The fixed bit length can vary (like 32-bit or 64-bit or 128-bit or 256-bit) depending on the hash function which is being used. The fixed-length output is called a hash. This hash is also the cryptographic byproduct of a hash algorithm. We can understand it from the following diagram.



The hash algorithm has certain unique properties:

1. It produces a unique output (or hash).

2. It is a one-way function.

In the context of cryptocurrencies like Bitcoin, the blockchain uses this cryptographic hash function's properties in its consensus mechanism. A cryptographic hash is a digest or digital fingerprints of a certain amount of data. In cryptographic hash functions, the transactions are taken as an input and run through a hashing algorithm which gives an output of a fixed size.

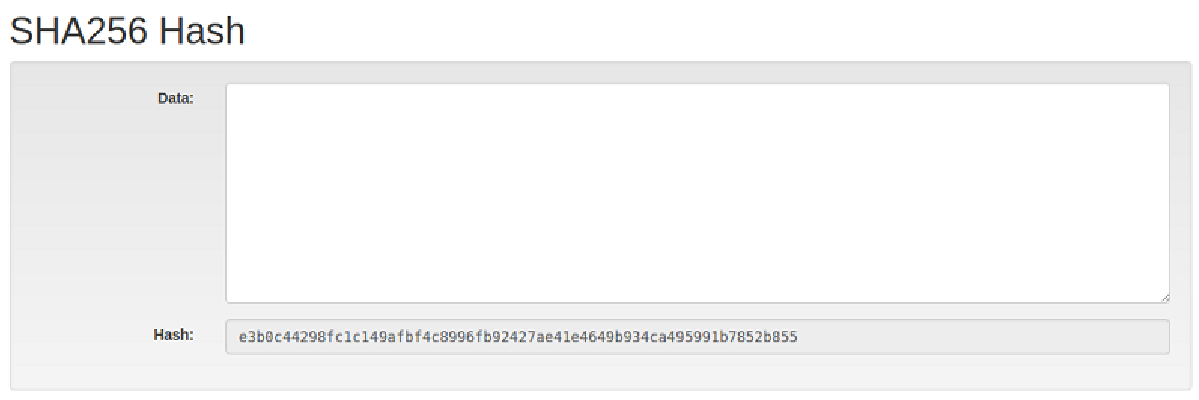
SHA-256

A Bitcoin's blockchain uses SHA-256 (Secure Hash Algorithm) hashing algorithm. In 2001, SHA-256 Hashing algorithm was developed by the National Security Agency (NSA) in the USA.

How does the hashing process works?

For this hash function, we are going to use a program developed by Anders Brownworth. This program can be found in the below link.

Anders Brownworth Hash Program: https://anders.com/blockchain/hash.html


SHA256 Hash

Data:

Hash: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

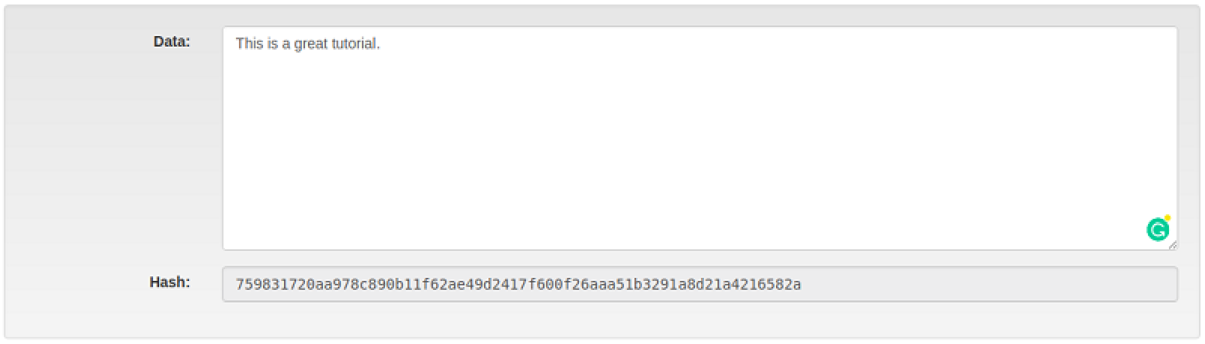
If we type any character in the data section, we will observe its corresponding cryptographic hash in the hash section.

For example: We have type in data section: **This is a great tutorial**.

It will generate the corresponding Hash:

1. 759831720aa978c890b11f62ae49d2417f600f26aaa51b3291a8d21a4216582a

SHA256 Hash


Data: This is a great tutorial.

Hash: 759831720aa978c890b11f62ae49d2417f600f26aaa51b3291a8d21a4216582a

Now if we change the text: "**This is a great tutorial**." To "**this is a great tutorial**."

You will find the corresponding Hash:

1. 4bc35380792eb7884df411ade1fa5fc3e82ab2da76f76dc83e1baecf48d60018

In the above, you can see that we have changed only the first character case sentence from capital "T" to small "t" and it will change the whole Hash value.

Note: If we write the same text again in a data section, it will always give the same output. It is because you are creating a message digest of that one's specific amount of data.

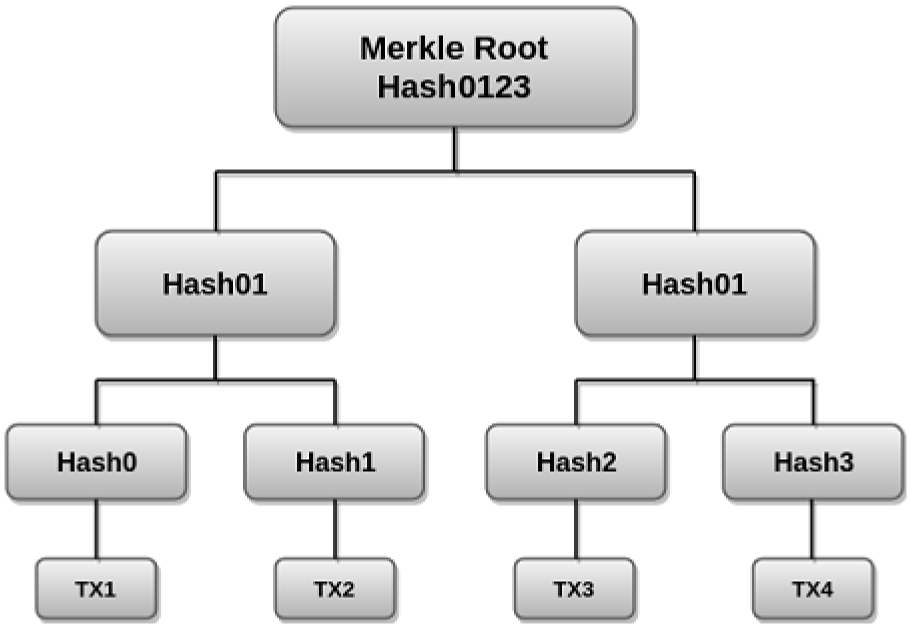
Since the Hash function is a one-way function, there is no way to get back entire text from the generated hash. This is different from traditional cryptographic functions like encryption where you can encrypt something using the key and by using decryption, you can decrypt the message to its original form.

4) Explain merkle tree.
   It is another useful data structure being used in blockchain solutions such as Bitcoin. Merkle trees are constructed by hashing paired data (usually transactions at the leaf level), then again hashing the hashed outputs all the way up to the root node, called the Merkle root. Like any other tree, it is constructed bottom-up. In Bitcoin, the leaves are always transactions of a single block in a blockchain.

A Merkle tree stores all the transactions in a block by producing a digital fingerprint of the entire set of transactions. It allows the user to verify whether a transaction can be included in a block or not.

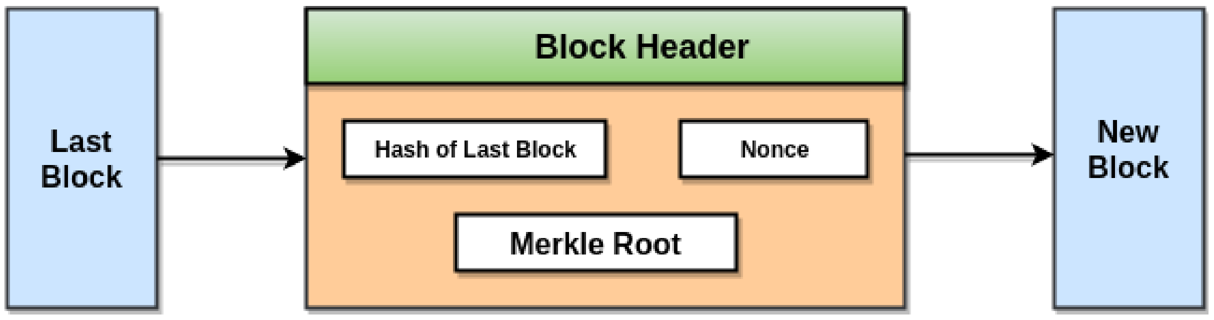
Merkle trees are created by repeatedly calculating hashing pairs of nodes until there is only one hash left. This hash is called the Merkle Root, or the Root Hash. The Merkle Trees are constructed in a bottom-up approach.

Every leaf node is a hash of transactional data, and the non-leaf node is a hash of its previous hashes. Merkle trees are in a binary tree, so it requires an even number of leaf nodes. If there is an odd number of transactions, the last hash will be duplicated once to create an even number of leaf nodes.



The above example is the most common and simple form of a Merkle tree, i.e., **Binary Merkle Tree**. There are four transactions in a block: **TX1**, **TX2**, **TX3**, and **TX4**. Here you can see, there is a top hash which is the hash of the entire tree, known as the **Root Hash**, or the **Merkle Root**. Each of these is repeatedly hashed, and stored in each leaf node, resulting in Hash 0, 1, 2, and 3. Consecutive pairs of leaf nodes are then summarized in a parent node by hashing **Hash0** and **Hash1**, resulting in **Hash01**, and separately hashing **Hash2** and **Hash3**, resulting in **Hash23**. The two hashes (**Hash01** and **Hash23**) are then hashed again to produce the Root Hash or the Merkle Root.

Merkle Root is stored in the **block header**. The block header is the part of the bitcoin block which gets hash in the process of mining. It contains the hash of the last block, a Nonce, and the Root Hash of all the transactions in the current block in a Merkle Tree. So having the Merkle root in block header makes the transaction **tamper-proof**. As this Root Hash includes the hashes of all the transactions within the block, these transactions may result in saving the disk space.



The Merkle Tree maintains the **integrity** of the data. If any single detail of transactions or order of the transaction's changes, then these changes reflected in the hash of that transaction. This change would cascade up the Merkle Tree to the Merkle Root, changing the value of the Merkle root and thus invalidating the block. So everyone can see that Merkle tree allows for a quick and simple test of whether a specific transaction is included in the set or not.

Merkle trees have three benefits:

- It provides a means to maintain the integrity and validity of data.
- It helps in saving the memory or disk space as the proofs, computationally easy and fast.
- Their proofs and management require tiny amounts of information to be transmitted across networks.

5) What is Bitcoin. With the neat diagram explain block structure of bitcoin blockchain.

   Bitcoin is a decentralized cryptocurrency that is not limited to any nation and is a global currency.

Block Structure

The block structure of a Bitcoin blockchain is fixed for all blocks and has specific fields with their corresponding required data. Take a look at Figure 3-5, a birds-eye view of the entire block structure, and then we will learn more about the individual fields later in this chapter.

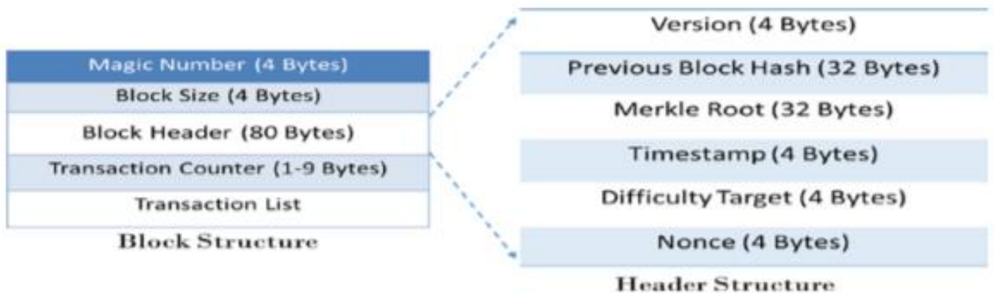


Figure 3-5. *Block structure of Bitcoin blockchain*

A typical block structure appears as shown in Table 3-1.

Table 3-1. *Block Structure*

Field	**Size**	**Description**
Magic Number	4 bytes	It has a fixed value 0xD9B4BEF9, which indicates the start of the block and also that the block is from the mainnet or the production network.
Block Size	4 bytes	This indicates the size of the block. The original Bitcoin blocks are of 1MB and there is a newer version of Bitcoin called "Bitcoin Cash" whose block size is 2MB.
Block Header	80 bytes	It comprises much information such as Previous Block's hash, Nonce, Merkle Root, and many more.

(*continued*)

Table 3-1. *(continued)*

Field	Size	Description
Transaction Counter	1–9 bytes (variable length)	It indicates total number of transactions that are included within the block. Not every transaction is of the same size, and there is a variable number of transactions in every block.
Transaction List	Variable in number but fixed in size	It lists all the transactions that are taking place in a given block. Depending on block size (whether 1MB or 2MB), this field occupies the remaining space in a block.

Let us now zoom in (Table 3-2) to the "Block Header" section of the blocks and learn the various different fields that it maintains.

Table 3-2. *Block Header Components*

Field	Size	Description
Version	4 bytes	It indicates the version number of Bitcoin protocol. Ideally each node running Bitcoin protocol should have the same version number.
Previous Block Hash	32 bytes	It contains the hash of the block header of the previous block in the chain. When all the fields in the previous block header are combined and hashed with SHA256 algorithm, it produces a 256-bit result, which is 32 bytes.

(continued)

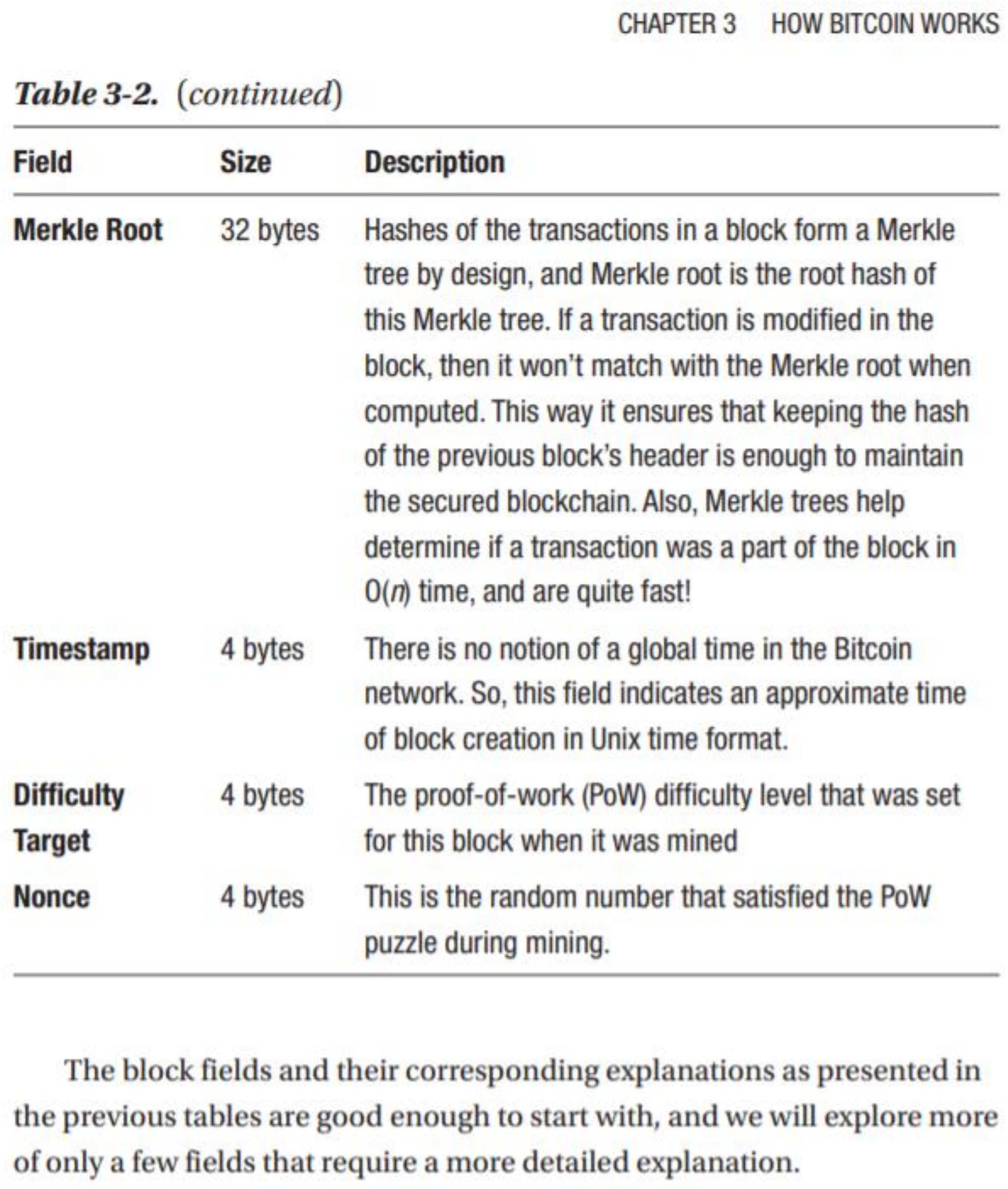

CHAPTER 3 HOW BITCOIN WORKS

Table 3-2. (*continued*)

Field	Size	Description
Merkle Root	32 bytes	Hashes of the transactions in a block form a Merkle tree by design, and Merkle root is the root hash of this Merkle tree. If a transaction is modified in the block, then it won't match with the Merkle root when computed. This way it ensures that keeping the hash of the previous block's header is enough to maintain the secured blockchain. Also, Merkle trees help determine if a transaction was a part of the block in O(*n*) time, and are quite fast!
Timestamp	4 bytes	There is no notion of a global time in the Bitcoin network. So, this field indicates an approximate time of block creation in Unix time format.
Difficulty Target	4 bytes	The proof-of-work (PoW) difficulty level that was set for this block when it was mined
Nonce	4 bytes	This is the random number that satisfied the PoW puzzle during mining.

The block fields and their corresponding explanations as presented in the previous tables are good enough to start with, and we will explore more of only a few fields that require a more detailed explanation.

6) Explain the following:

i. difficulty target

The difficulty is a measure of how difficult it is to mine a Bitcoin block, or in more technical terms, to find a hash below a given target. A high difficulty means that it will take more computing power to mine the same number of blocks, making the network more secure against attacks. The difficulty adjustment is directly related to the total estimated mining power estimated in the Total Hash Rate (TH/s) chart.

ii. Genesis block

The very first block as you can see in the following code, the block-0, is called the genesis block. Remember that the genesis block has to be hardcoded into the blockchain applications and so is the case with Bitcoin. You can consider it as a special block because it does not contain any reference to the previous blocks. The Bitcoin's genesis block was created in 2009 when it was launched. If you open the Bitcoin Core, specifically the file chainparams. cpp, you will see how the genesis block is statically encoded. Using a command line reference to Bitcoin Core, you can get the same information by querying with the hash of the genesis block as shown below:

iii. Bitcoin block

What Is a Block (Bitcoin Block)? Blocks are **files where data pertaining to the Bitcoin network are permanently recorded**. A block records some or all of the most recent Bitcoin transactions that have not yet entered any prior blocks. Thus, a block is like a page of a ledger or record book.

7) Explain bitcoin scripts.

Bitcoin Scripts We learned about Bitcoin transactions in previous sections at a high level. In this section we will delve deep into the actual programming constructs that make the transactions happen. Every Bitcoin transactions' input and output are embedded with scripts. Bitcoin scripts are stack based, Chapter 3 How Bitcoin Works 196 which we will see shortly, and are evaluated from left to right. Remember that Bitcoin scripts are not Turing-complete, so you cannot really do anything and everything that is possible through other Turing-complete languages such as C, C++, or Java, etc. There are no concepts of loops in Bitcoin script, hence the execution time for scripts is not variable and is proportional to the number of instructions. This means that the scripts execute in a limited amount of time and there is no possibility for them to get stuck inside a loop. Also, most importantly, the scripts definitely terminate. Now that we know a little about the scripts, where do they run? Whenever transactions are made, whether programmatically, or through a wallet software or any other program, the scripts are injected inside the transactions and it is the work of the miners to run those scripts while mining. The purpose of Bitcoin scripts is to enable the network nodes to ensure the available funds are claimed and spent only by the entitled parties that really own them.

8) With diagram explain merkle root a block header of SPV.

SPVs

Bitcoin design has this nice concept of Simple Payment Verification(SPV) nodes that can be used to verify transactions without running full nodes. The way SPVs work is that they download only the header of all the blocks during the initial syncing to the Bitcoin network. In Bitcoin, the block headers are of 80 bytes each, and downloading all the headers is not much and ranges to a few MBs in total.

The purpose of SPVs is to provide a mechanism to verify that a particular transaction was in a block in a blockchain without requiring the entire blockchain data. Every block header has the Merkle root, which is the block hash. We know that every transaction has a hash and that transaction hash can be linked to the block hash using the Merkle tree proof which we discussed in the previous chapter. All the transactions
in a block form the Merkle leafs and the block hash forms the Merkle root. The beauty of the Merkle tree is that only a small part of the block is needed to prove that a transaction was actually a part of the block. So, to confirm a transaction an SPV does two things. First, it checks the Merkle tree proof for a transaction to accertain it is a part of the block and second, if that block is a part of the main chain or not; and there should be at least six more blocks created after it to confirm it is a part of the longest chain. Figure 3-26 depicts this process.

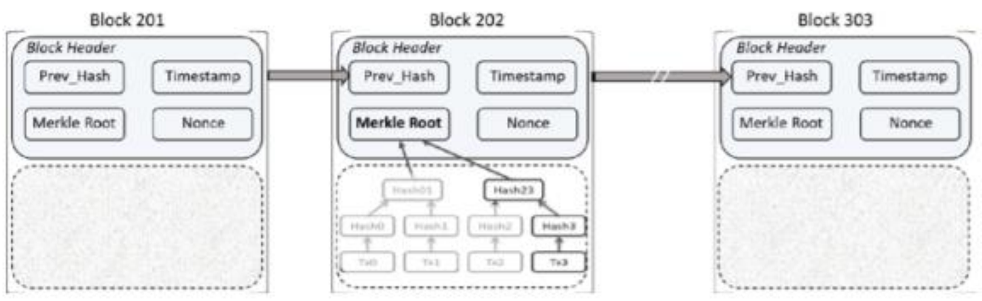


Figure 3-26. *Merkle root in block header of SPV*

Let us dig deeper into the technicality of how SPVs really work in verifying transactions. At a high level, it takes the following steps:

- To the peers an SPV is connected to, it establishes Bloom filters with many of them and ideally not just to one peer, because there could be a chance for that peer to perform denial of service or cheat. The purpose of Bloom filters is to match only the transactions an SPV is interested in, and not the rest in a block without revealing which addresses or keys the SPV is interested in.
- Peers send back the relevant transactions in a *merkleblock* message that contains the Merkle root and the Merkle path to the transaction of interest as shown in the figure above. The *merkleblock* message sze is in a few kB and quite efficient.
- It is then easy for the SPVs to verify if a transaction truly belongs to a block in the blockchain.
- Once the transaction is verified, the next step is to check if that Block is actually a part of the true longest blockchain.

The following (Figure 3-27) represents this SPV communication steps with its peers.

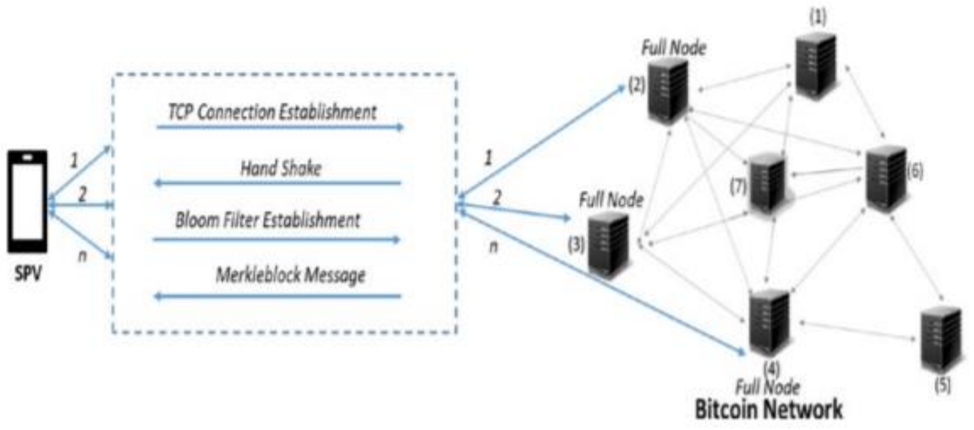


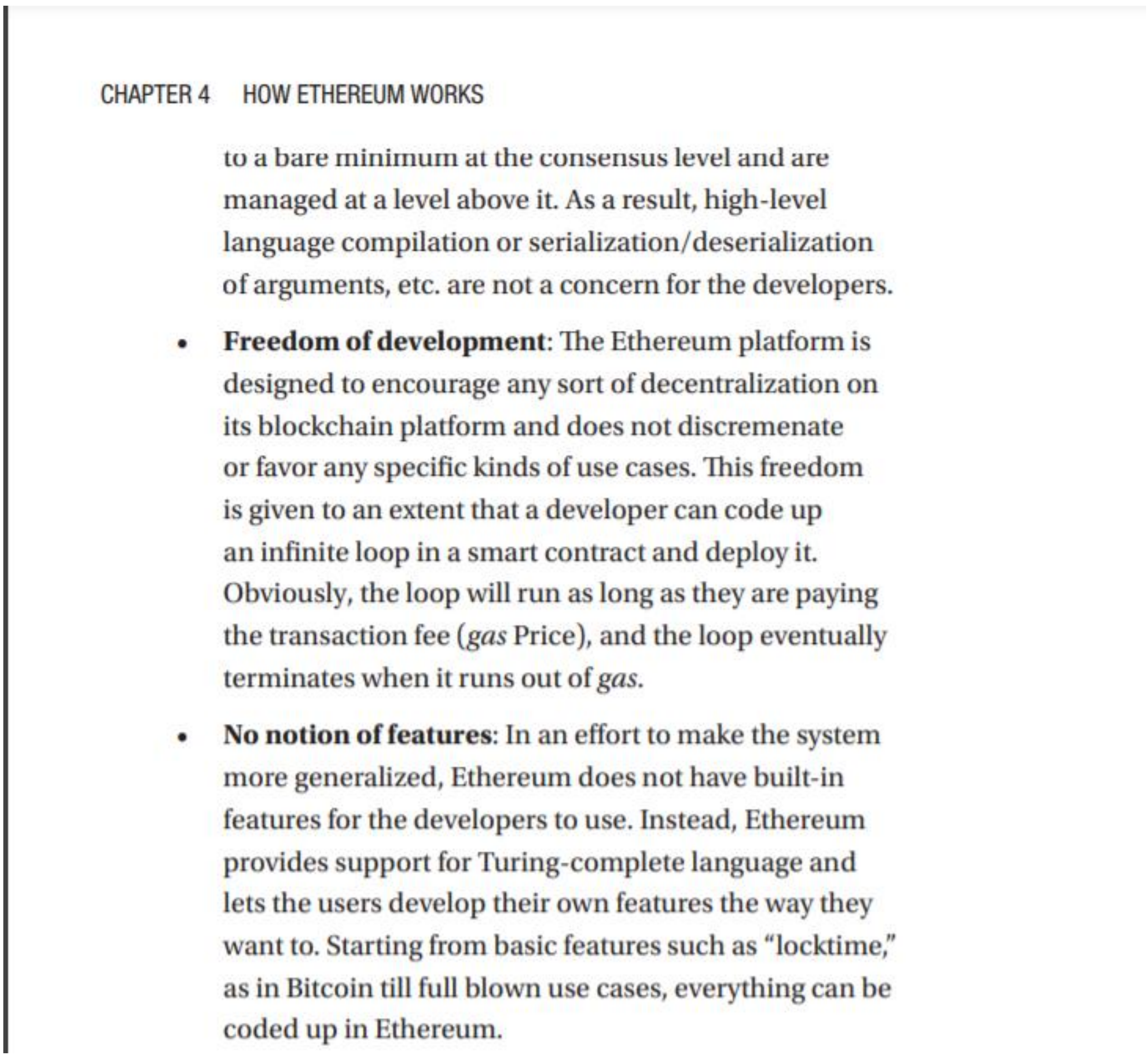
Figure 3-27. *SPV communication mechanism with the Bitcoin network*

9) What is etherium and explain design philosophy of etherium.

Design Philosophy of Ethereum

Ethereum borrows many concepts from Bitcoin Core as it stood the test of time, but is designed with a different philosophy. Ethereum development has been done following certain principles as follows:

- **Simplistic design**: The Ethereum blockchain is designed to be as simple as possible so that it is easy to understand and develop decentralized applications on. The complexities in the implementation are kept



to a bare minimum at the consensus level and are managed at a level above it. As a result, high-level language compilation or serialization/deserialization of arguments, etc. are not a concern for the developers.

- **Freedom of development**: The Ethereum platform is designed to encourage any sort of decentralization on its blockchain platform and does not discremenate or favor any specific kinds of use cases. This freedom is given to an extent that a developer can code up an infinite loop in a smart contract and deploy it. Obviously, the loop will run as long as they are paying the transaction fee (*gas* Price), and the loop eventually terminates when it runs out of *gas*.
- **No notion of features**: In an effort to make the system more generalized, Ethereum does not have built-in features for the developers to use. Instead, Ethereum provides support for Turing-complete language and lets the users develop their own features the way they want to. Starting from basic features such as "locktime," as in Bitcoin till full blown use cases, everything can be coded up in Ethereum.

10) Explain etherium blockchain.

Ethereum Blockchain

The Ethereum blockchain data structure is pretty similar to that of Bitcoin's, except that there is a lot more information contained in the block header to make it more robust and help maintain the state properly. We will learn more about the Ethereum states in the following sections. Let us focus more on the blockchain data structure and the header in this section. In Bitcoins, there was only one Merkle root in the block header for all the transactions in a block. In Ethereum, there are two more Merkle roots, so there are three Merkle roots in total as follows:

- **stateRoot**: It helps maintain the global state.
- **transactionsRoot**: It is to track and ensure integrity of all the transactions in a block, similar to Bitcoin's Merkle root.
- **receiptsRoot**: It is the root hash of the receipts *trie* corresponding to the transactions in a block

We will take a look at these Merkle roots in their respective sections of block header information. For better comprehension, take a look at Figure 4-2.

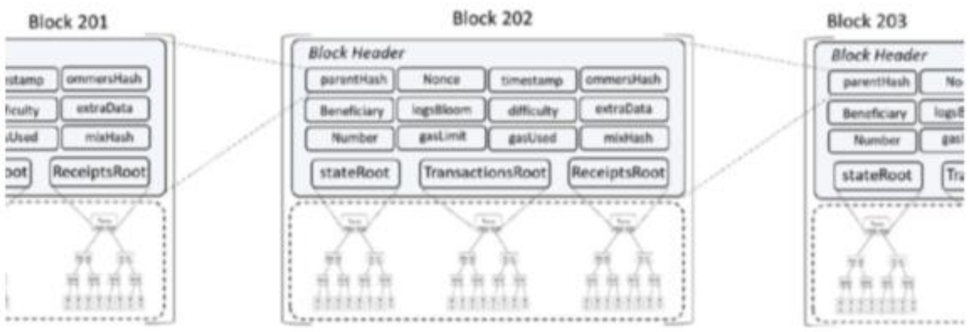


Figure 4-2. *The blockchain data structure of Ethereum*

Every block usually comprises block header, transactions list, uncles list, and optional extraData. Let us now take a look at the header fields to understand what they mean and their purpose for being in the header. While you do so, keep in mind that there could be slight variants of these names in different places, or the order in which they are presenbted could be different in different places. We suggest that you build a proper understanding of these fields so that any different terminology that you might come across won't bother you much.

Section-1: Block metadata

- parentHash: Keccak 256-bit hash of the parent block's header, like that of Bitcoin's style
- timestamp: The Unix timestamp current block
- number: Block number of the current block
- Beneficiary: The 160-bit address of "author" account responsible for creating the current block to which all the fees from successfully mining a block are collected

Section-2: Data references

- transactionsRoot: The Keccak 256-bit root hash (Merkle root) of the transactions trie populated with all the transactions in this block
- ommersHash: It is otherwise known as "uncleHash." It is the hash of the uncles segment of the block, i.e., Keccak 256-bit hash of the ommers list portion of this block (blocks that are known to have a parent equal to the present block's parent's parent).
- extraData: Arbitrary byte array containing data relevant to this block. The size of this data is limited to 32 bytes (256-bits). As of this writing, there is a possibility that

this field might become “extraDataHash”, which will point to the “extraData” contained inside the block. extraData could be raw data, charged at the same amount of *gas* as that of transaction data.

Section-3: Transaction execution information

- stateRoot: The Keccak 256-bit root hash (Merkle root) of the final state after validating and executing all transactions of this block
- receiptsRoot: The Keccak 256-bit root hash (Merkle root) of the receipts trie populated with the recipients of each transaction in this block
- logBloom: The accumulated Bloom filter for each of the transactions’ receipts’ Blooms, i.e., the “OR” of all of the Blooms for the transactions in the block
- *gasUsed*: The total amount of *gas* used through each of the transactions in this block
- *gasLimit*: The maximum amount of *gas* that this block may utilise (dynamic value depending on the activity in the network)

Section-4: Consensus-subsystem information

- difficulty: The difficulty limit for this block calculated from the previous block’s difficulty and timestamp
- mixHash: The 256-bits mix hash combined with the ‘nonce’ for the PoW of this block
- nonce: The nonce is a 64-bit hash that is combined with mixHash and can be used as a PoW verification.