Principles of Blockchains
Princeton University,
Professor: Pramod Viswanath

Notes - complete edition

July 21, 2023

Contents

Chapter 1

Introduction

1.1 What are blockchains

Blockchain is a bit different with other subjects, in the sense that it needs to be first defined. It is remarkable and so many people try to define it in their own way. Our definition of blockchain is as follows :

Definition 1.1.1: Blockchain

Blockchains are the technology underlying **decentralized** **digital trust** **platforms**.

Decentralized system

A decentralized system is one where no single entity (person/company) is responsible for the smooth operation of the system. Indeed, blockchains are peer-to-peer systems, where each peer has the same prescribed behavior; no peer is unique. Peers communicate with each other by exchanging messages. Beyond this message exchange, peers function independently of one another.

Trust

Trust is a medium that drives human interactions. Human success is based on flexible cooperation in large numbers. This requires trust. The underlying mechanism of trust dictates how large the size of human cooperation can get.

Types of trust

- **Tribal Trust :**
  This is the traditional version of trust. People who shared similarities in language, generics compositions and customs, organized themselves around tribal societies. It was not very scaled in number or geographical place and was rather local.

- **Institutional Trust :**
  Since the end of WWII, the dominant human organization has been institutional. The underlying mechanism of trust is a set of laws and transparent and rigorous enforcement of the law. "no one is above the law".
  This notion of trust has enabled human societies to cooperate on a global scale. It was extended in every aspect of human activity, travel, commerce, communication.
  However, the drawback is that the institutions are centralized, and the power to enforce and promulgate the laws is concentrated in the hands of a few individuals. The concentration of power leads to unintended consequences, including corruptibility, autocratic tendencies, and rent-seeking.

- **Distributed trust :**
  The siren song of blockchains is that the scaling and flexibility of institutional trust are possible without its negative aspects, i.e., the creation of a decentralized trust system. This is where blockchains play a significant role.

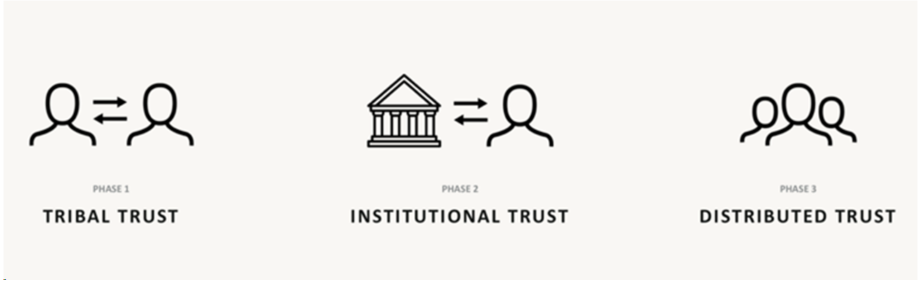
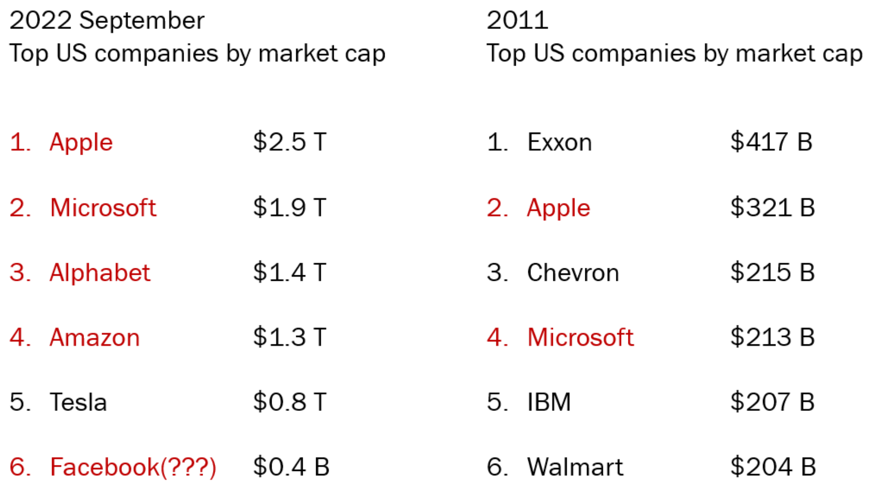


Figure 1.1: Evolution of Trust over human history

Blockchains are guaranteed to be secure even if some fraction of peers act maliciously. The only requirement is that sufficiently many peers operate according to the prescribed behavior. This is called the honest majority assumption. Thus, any user of a blockchain does not need to trust all peers in the system or even one particular peer. Instead, it just needs to trust that a majority of peers are honest. This is a key feature of blockchains that are not present in other peer-to-peer systems, or indeed, any other system.

Digital platforms

Digital platforms are marketplaces where suppliers and consumers come together and trade. A digital platform is one in which two (or more) parties interact, and some transaction takes place. There are many examples of digital platforms today. For example, Amazon/eBay is a platform for the exchange of goods. It provides a place where sellers can list goods that they are selling, and buyers can choose to buy any listed goods. Among other things, the platform ensures that the transaction takes place securely. It also handles disputes in transactions. Other examples of platforms are Uber/Lyft (for rides), AirBnB (for temporary accommodation), etc. Digital platforms have played a significant role in the global economy in the last decade.For the last decade and beyond, platform companies have been the dominant force in the economy. Here we put the top five, six companies by market cap. They're all platform companies. The aforementioned digital platforms are not truly decentralized in the sense, they do not offer distributed trust. By using any platform, we implicitly trust the (single) company behind that platform to handle transactions faithfully. The platform holds a lot of power in arbitrating disputes. It can also arbitrarily decide the fees to charge for the service. (Of course, the reputation/popularity of the platform is at stake, which prevents it from behaving arbitrarily).



2022 September Top US companies by market cap		2011 Top US companies by market cap	
1. Apple	$2.5 T	1. Exxon	$417 B
2. Microsoft	$1.9 T	2. Apple	$321 B
3. Alphabet	$1.4 T	3. Chevron	$215 B
4. Amazon	$1.3 T	4. Microsoft	$213 B
5. Tesla	$0.8 T	5. IBM	$207 B
6. Facebook(???)	$0.4 B	6. Walmart	$204 B

Figure 1.2: Evolution of Trust over human history

Blockchains hold the promise to decentralize the digital platforms we see today. From a user's perspective,

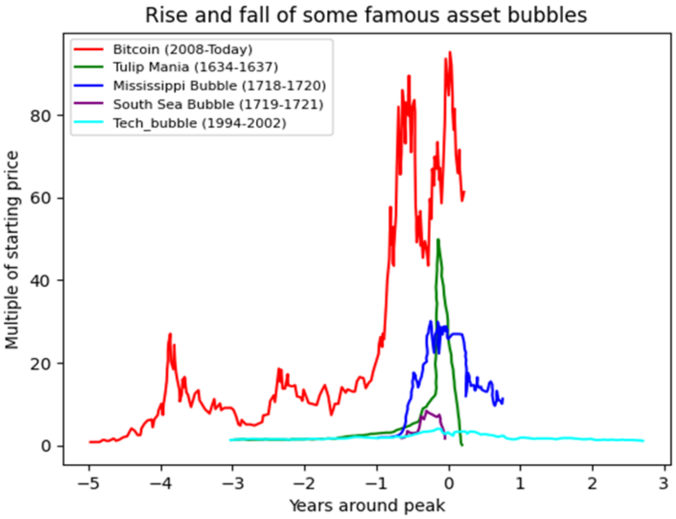


Figure 1.3: Bitcoin and Bubbles

the platform's functionality will be identical. However, the platform will no longer be operated by a single company but rather a multitude of small stakeholders, each running the same blockchain code. The 'trust' factor in the system becomes distributed.

1.2 Bitcoin as the first blockchain

The term blockchain was introduced in late 2008 with the advent of Bitcoin. The inventor of bitcoin is Satoshi Nakamoto, a pseudonym for the person or persons who developed the cryptocurrency, authored the bitcoin white paper, and created and deployed the original bitcoin software. Bitcoin is a cryptocurrency: a decentralized, digital payment platform. As such, cryptocurrencies are one of the simplest applications of blockchains. Today, there is a multitude of blockchain designs for cryptocurrencies and other applications. However, they all retain many of the core design components that were introduced in Bitcoin. Thus, the term 'blockchain' has remained in all of them.

Bitcoin is one among many attempts in history to create a decentralized, digital payment platform (the term 'currency' is to be thought of as a 'token' that is used on this payment platform). Unlike all previous attempts, Bitcoin has stood the test of time. Its popularity, which can be measured by its price in dollars, has grown many-fold over the twelve years since its introduction.

It may seem that the fall and rise in the price of Bitcoin is a bubble, but based on Figure 1.3, the red graph that shows the changes in the price of Bitcoin says that the price of Bitcoin has gone up and down, but other bubbles burst at once and their prices do not rise much. This shows that Bitcoin is stable. Maybe it can be said that this is the mother of all bubbles. A major reason behind the popularity of Bitcoin is its strong security property, coupled with its truly decentralized nature. It is now well understood that as long as 51% of the peers in Bitcoin are honest, the system is secure (the exact conditions are slightly different, but this suffices for the moment). This has been shown theoretically and has also been borne out in practice. Moreover, anyone can freely join and leave the Bitcoin system at any time; the system is **'permissionless'**.

Some properties of Bitcoin and its performance are as follows:

1. **Security**: We will discuss what does it mean that bitcoin is secure in future notes but generally bitcoin is secure as it is almost always accessible and it have never been broken.

2. **Transaction throughputs**: Bitcoin has transaction of 7 tx/s which is noticeably low in amount. (See figure 1.4)

3. **Confirmation Latency** : Bitcoin has a very slow latency which can take hours before something gets confirmed.

4. **Energy consumption**: Bitcoin and it's mining is famous for its large energy consumption. It's that of a medium sized country.

5. **Computation and storage**: Mining bitcoin requires huge resources of computing and storage.

6. **Communication**: Communication requirements are also fairly high. Everybody's transmitting everything and every possible message.

Despite its benefits, there are major drawbacks of Bitcoin, which impact its viability. Some issues can be categorized as scaling issues. For example, the system can only process about seven transactions per second. In contrast, Visa (a centralized digital payment mechanism) processes 50,000 transactions per second. If all people in the world are to switch from Visa to Bitcoin, the system must 'scale' its transaction throughput. Other issues are a lack of desirable properties. For example, if the network is disrupted, transactions that are once 'confirmed' can be reverted. (We say that Bitcoin does not offer 'finality' in confirming transactions).

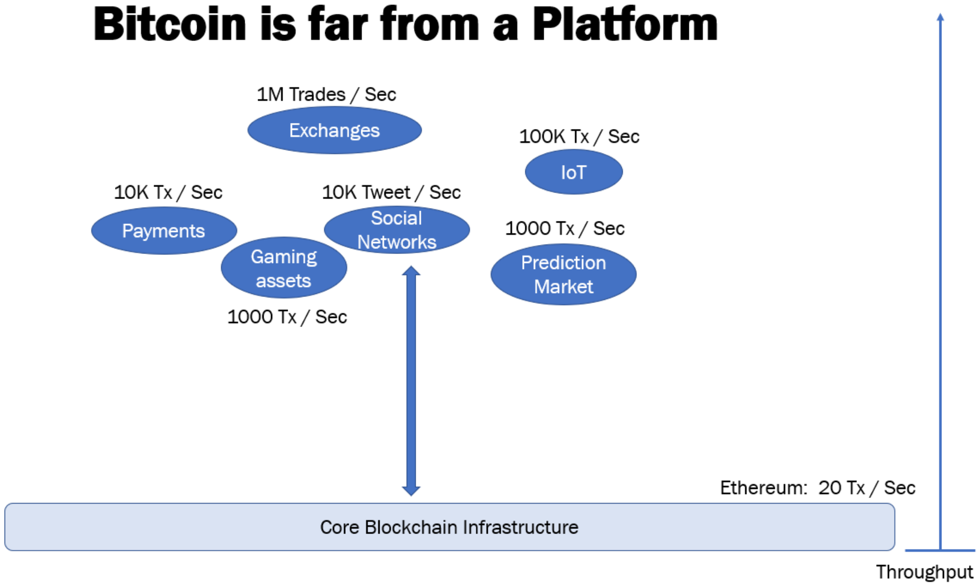


Figure 1.4: Evolution of Trust over human history

Technical components

- **decentralized computer**:
  we are looking at sort of libraries, basic libraries that we are used to, which includes both networking libraries and cryptographic libraries. It mainly contains the followings:
  - Cryptographic data structures
  - Disk I/O, memory/cache and Database management
  - Operating systems
  - Peer to peer networking
  - Consensus and distributed algorithms
- **Virtual machine**:
  - **Reduced instruction set, incentives :**
    A virtual machine uses a simplified instruction set. It also has a decentralized structure that rewards each participant for their contribution, whether it is storage, computation, or data. This creates incentives at the instruction level, similar to how ants benefit from their collective work.

- **General purpose programming language** :
  A programming language can be built on top of this system.

Technical challenges

- **Permissionless**:
  One of the main goals of designing a decentralized system is to achieve permissionless participation, where anyone can join and contribute to the network without requiring any authorization or intermediaries. However, this also poses a technical challenge, as the system needs to prevent spam and malicious behavior from adversaries who may try to disrupt or compromise the network.
- **Dynamic availability and safety**:
  Another goal is to ensure dynamic availability and safety, where the network can reach consensus and maintain its integrity even with changing and unpredictable participation of nodes. This also requires a robust mechanism to deal with potential attacks from adversaries who may try to manipulate or censor the network.
- **Cryptography**:
  Cryptography provides some basic tools to address both of these design goals, such as digital signatures, hash functions, encryption, and zero-knowledge proofs. However, cryptography alone is not enough, as adversaries may also use sophisticated techniques to counter or exploit these tools. As the famous quote goes, ”The trouble is, the other side can do magic too, Prime Minister.” Therefore, designing a decentralized system requires a careful balance of cryptography, incentives, game theory, and distributed algorithms.

1.3 What this course is about

This course covers the **fundamental design principles of blockchains**. We aim to look at good blockchain designs as a full computer rather than isolated elements (we are not going to look only at cryptography parts or algorithms part, ...). Basic background in algorithms, probability system programming skills (C/C++) and maturity with nearly all aspects of computer science are required and will be used.

The lectures are divided into three modules:

1. **Understanding Bitcoin**
   The first six lectures cover the complete design of Bitcoin. At the end of this module, you will be able to understand the system as a whole and how it functions as a secure, decentralized payment platform. We will introduce various terms pertaining to Bitcoin and cryptocurrencies (e.g., UTXOs, hash pointers, longest chain rule) and cryptography on a “need to know basis.” We will study some attacks on Bitcoin and under what conditions Bitcoin is secure under those attacks.
2. **Scaling Bitcoin**
   The second module aims to improve the throughput, latency, and resource usage (energy, compute, storage, and communication needs) of Bitcoin while maintaining the same security levels. The goal is to work within the overall architecture of Bitcoin itself and systematically improve various design elements.
3. **Beyond Bitcoin**
   The third module covers alternate blockchain protocols that offer properties that are simply not present in Bitcoin. These include accountability, resistance to 51% adversarial attacks, transaction finality, and privacy (the exact meaning of these terms will be made clear later on). We also see how these properties can be combined with Bitcoin-like blockchains via the notion of a ‘finality gadget.’ This gives us a blockchain with many desirable properties.

This course emphasizes the implementation of blockchains (in software). Thus, the bulk of the evaluation will be based on two projects. The first project involves implementing a Bitcoin client. This will make use of all the concepts learned in the first module of the course. The second project will involve implementing a finality gadget of your choice on top of Bitcoin (from the third module). Rust will be the programming language that is used.

1.4 What is Rust?

Rust is a compiled language such as C and C++ and build for reliability. In Rust we have runtime and compile time safety together and it helps having memory and thread safety. This is very important in blockchain. In general, when a language or model is agreed upon in the blockchain, changing it is indistinguishable from a security attack. Therefore, when the differential method is done, the blockchain cannot be changed or is very difficult to change. Therefore, unlike other applications in the blockchain, it is not easy to add new code and new changes.

For example, the changes that took place in Ethereum took several years because, in addition to the above, it was necessary for all participants to vote, take a stand and move towards the changes. Therefore, the blockchain program must be well written, and for this reason, rust is used.

Chapter 2

Blockchains as Cryptographic Data Structures

Since now we have got familiar with blockchains but now we discuss blockchains in another narrow point of view. We are going to look at blockchains as a data structure with cryptographic properties. We will see what properties this data structure satisfies and that's the key to a blockchain. Remember the two keywords for blockchains, decentralization, and trust. We should be able to link back these properties to trust, and perhaps decentralization.

We will discuss three basic cryptographic primitives :

1. Cryptographic Hash Functions
2. Hash Accumulators, Merkle trees
3. Digital Signatures

The first two primitives are put together to create blockchain that provides trust. The third primitive signatures allow us to go towards decentralization.

2.1 Hash Functions

A hash function is a function that converts a binary string of arbitrary length to a binary string of fixed length.For any piece of data x, the output of the hash function is denoted by $H(x)$ and is called the hash of x.

Example 2.1.1 (Division hashing)

$$y = x \mod 2^{256}$$

Think of x as a binary sequence. This hash function modulate the input with 2^{256} and the output output is a 256 bit sequence, which is the remainder. dividing by a large number of the remainder is uniform between 0 and $2^{256} - 1$. It's a simple deterministic hash function which can be simply computed.

A **collision** happens when there are two inputs, x and x', which get mapped to the same hash, i.e., $H(x) = H(x')$. To minimize collisions, a hash function must distribute its output uniformly and as spread out as possible over the output. space. Therefor, a good hash function should have following properties:

1. Arbitrary sized inputs
2. Fixed size deterministic output
3. Efficiently computable
4. Minimize collisions

Hash functions are widely used not only in blockchains but also in databases, data mining, and information retrieval. They are beneficial for indexing lots of data, especially the ones that are not already numbers e.g. files, images and audio.

2.2 Cryptographic Hash Function

Cryptographic hash functions are quit the same as hash function but with two extra properties:

1. **adversarially collision resistance**
2. **one way function**

The hash of a file should be unique and hard to replicate. An adversary should not be able to create another file that has the same hash as the original one. The previously provided example (2.1) is not adversarially collision resistant since we can simply add 2^{256} to the input and that will have the same output.
In this example if we pick random inputs and compute the hash values, we'll find a collision with high probability long before examining $2^{256} + 1$ inputs. In fact, if we randomly choose just $2^{130} + 1$ inputs, it turns out there's a 99.8% chance that at least two of them are going to collide. The fact that we can find a collision by only examining roughly the square root of the number of possible outputs results from a phenomenon in probability known as the birthday paradox .

The output space of cryptographic hash functions must be large, or else one could easily find a hash collision by iterating over the output space. Typically, they are 128-bit strings, 256-bit strings, or longer.

The key feature of a cryptographic hash function is that it is easy to compute, but difficult to invert. This means that given a binary string x, it is easy to compute $y = H(x)$, but given an arbitrary y', it is difficult and it would take a really long timeto find any x' for which $H(x') = y'$.

SHA-256

Constructing a cryptographic hash function is not easy (in contrast to regular hash functions). The National Institute of Standards and Technology (NIST) sets a standard for these hash functions. One such function is **SHA-256**, where SHA stands for Secure Hash Algorithm, and 256 is the length of the output.
They are built using the Merkle–Damgård construction, from a one-way compression function.

A hash of a certain value acts as a commitment for that value. One can broadcast the hash of the value instead of the value itself. Due to the collision resistance property, one cannot generate an alternate value that matches the same hash value; one is committed to the original value. Thus, publishing the hash of a value is like writing it on a piece of paper and placing it in a sealed envelope.

2.3 Hash Pointer and chains of block

A hash function can also be used as a pointer to certain values when these are stored in a hash table. Note that a hash pointer is nothing more than a hash; the term alludes to the fact that the hash is being used as a pointer. Hash pointer retrieve information and verify the information has not changed. Hash pointers can also be used to build related data structures. Crucially useful for blockchains. In fact, blockchain itself is a hash pointer-based data structure.

In the context of blockchains, a block is a data type that contains a particular header field, called the **hash pointer**, and some **data**.
Using hash pointer, we build a linked list using hash pointers (See figure 2.1). We're going to call this data structure a block chain . Whereas as in a regular linked list where you have a series of blocks, each block has data as well as a pointer to the previous block in the list, in a block chain the previous block pointer will be replaced with a hash pointer. So each block not only tells us where the value of the previous block was, but it also contains a digest of that value that allows us to verify that the value hasn't changed. We store the head of the list, which is just a regular hash-pointer that points to the most recent data block.

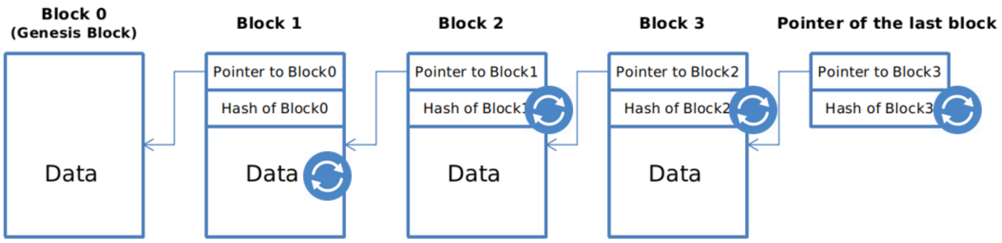


Figure 2.1: A block chain is a linked list that is built with hash pointers instead of pointers

A use case for a block chain is a tamper-evident log . That is, we want to build a log data structure that stores a bunch of data, and allows us to append data onto the end of the log. But if somebody alters data that is earlier in the log, we're going to detect it.

Blockchains are tamper-proof data structures, making them particularly useful in digital trust systems. The system consists of parties that keep their own hash tables as local copies of the data structure. A party can trace the lineage of any block (its parent, grandparent, etc.) using the hash pointers. If a party is missing a block in its hash table, it can ask its peers for the block by using the hash of the block (which it gets from the child block). It can then confirm that the block it gets from its peer is the right one (i.e., it has not been altered) by checking if its hash matches with what it has; this is essentially using the hash as a proof. Similarly, a party can verify if any part of the blockchain it receives has been changed or not.
A party may want to check if a specific data value is part of the blockchain. If the party has the whole blockchain and all its internal data, it can easily prove this. But for a party that only cares about one data value, this is too much work. To make this easier for practical blockchain systems, we use a Merkle tree data structure to store the data in each block. We explain this next.

2.4 Merkle tree

Another useful data structure that we can build using hash pointers is a binary tree. A binary tree with hash pointers is known as a Merkle tree , after its inventor Ralph Merkle. Suppose we have a number of blocks

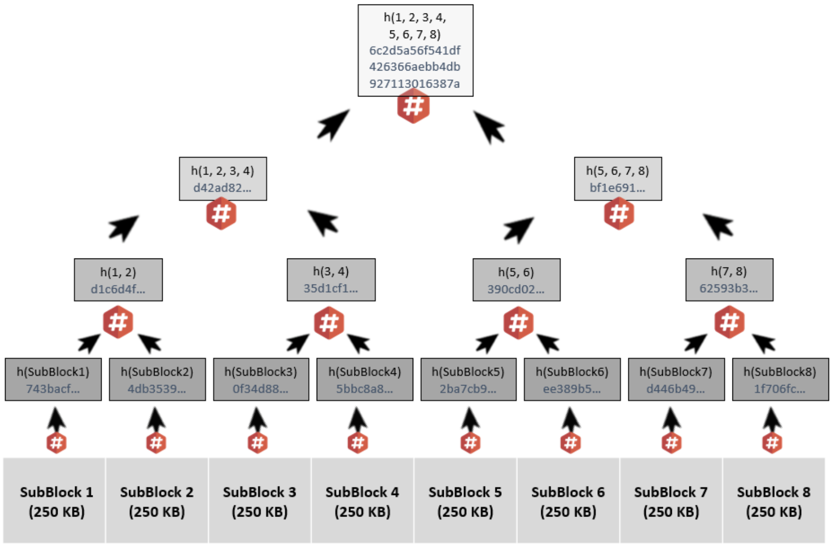


Figure 2.2: A Merkle tree

containing data. We group these data blocks into pairs of two, and then for each pair, we build a data structure that has two hash pointers, one to each of these blocks. We continue doing this until we reach a single block, the root of the tree.

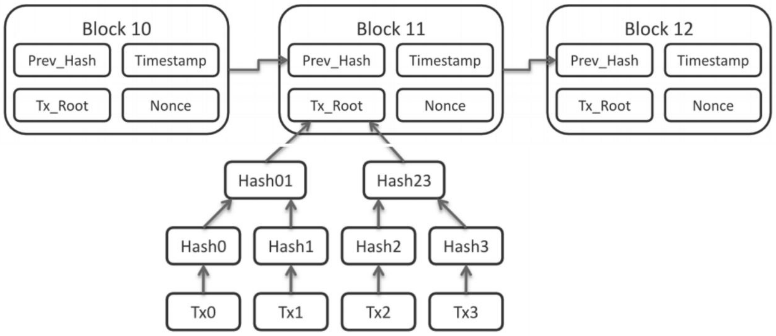


Figure 2.3: Blockchain with markle trees

As you see above, we have a blockchain and data of each block is stored as markle tree. We dont just keep the hash of data, we always keep the hash of markle. At this case header contains the hash of previous block and the root of markle tree.

Proof of membership

Say that someone wants to prove that a certain data block is a member of the Merkle Tree. As usual, we remember just the root. Then they need to show us this data block, and the blocks on the path from the data block to the root. We can ignore the rest of the tree, as the blocks on this path are enough to allow us to verify the hashes all the way up to the root of the tree.

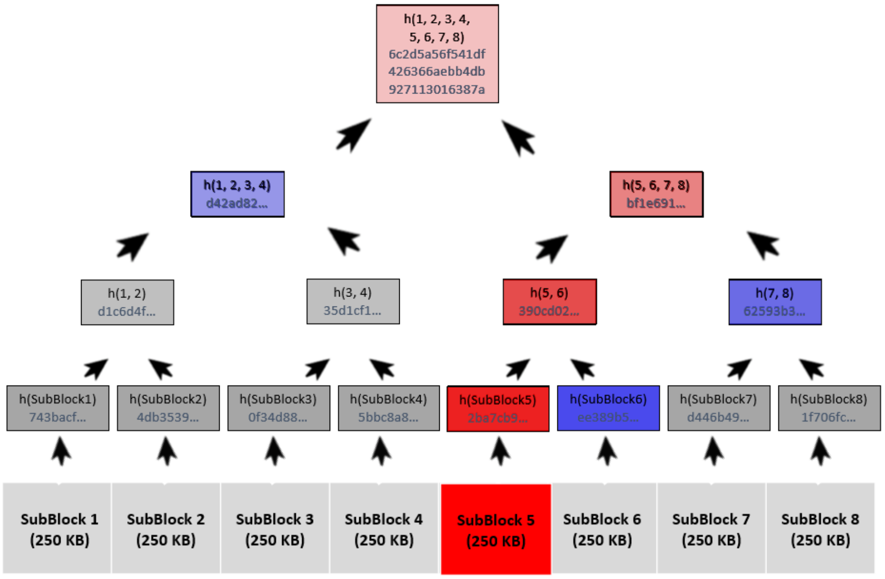


Figure 2.4: Finding a block in Merkle tree

If there are n nodes in the tree, only about log(n) items need to be shown. And since each step just requires computing the hash of the child block, it takes about log(n) time for us to verify it.

Proof of non-membership

. With a sorted Merkle tree, it becomes possible to verify non-membership in a logarithmic time and space. That is, we can prove that a particular block is not in the Merkle tree. And the way we do that is simply by showing a path to the item that's just before where the item in question would be and showing the path to the item that is just after where it would be. If these two items are consecutive in the tree, then this serves as a proof that the item in question is not included. For if it was included, it would need to be between the two items shown, but there is no space between them as they are consecutive.

2.5 Digital Signature

Digital signatures serve as the cryptographic equivalent of handwritten signatures. They enable anyone to verify the sender of a message. In a digital signature scheme, each user is provided with a pair of keys: a secret key known only to the user and a public key shared with everyone. To sign a message, a user employs their secret key, and the resulting signature is sent alongside the message. Another user can verify that the message indeed came from the purported sender by comparing the message and signature to the sender's public key. Thus, the public key becomes the user's identity in the system.

The security of a digital signature scheme lies in the inability of an adversary to forge signatures without knowledge of the corresponding secret key.

It is crucial that each message has a distinct signature. If the same signature could be used for multiple messages, an adversary could simply append the signature to other messages, rendering the scheme ineffective. Typically, users sign the hash of the message they intend to send, ensuring a constant-length bit string for both the signed object and the signature.

To ensure unforgeability, the signature length must be sufficient. Secure signature schemes are standardized by organizations such as NIST.

Bitcoin utilizes the Elliptic Curve Digital Signature Algorithm (ECDSA) as its signature scheme. Elliptic curves are employed in the signing process, but the details are beyond the scope of this discussion. Further information on ECDSA can be found on the Wikipedia page and associated links.

Digital signatures find their first application in upgrading a centralized blockchain system to a decentralized version. Each block in the blockchain is augmented with a digital signature, enabling different users to append blocks and identify themselves through their signatures.

In this decentralized architecture, there are three key questions to address:

1. Who are the eligible users allowed to participate, and how are they selected?

2. When and which block can a user append, and how can others verify this rule in a decentralized manner?

3. Where should a user append a block? In theory, a block can be attached to any other block as viewed by the user.

Blockchain designs vary in how they answer these three questions and the properties they possess. In the next lecture, we will explore how Bitcoin resolves these questions.

The second application of digital signatures involves providing user attribution to the data stored in blocks. While the data is considered an abstract digital entity, there are scenarios where attributing it to users is essential. For example, in cryptocurrencies, the data represents transactions that record the transfer of ownership of coins. By using signatures, user ownership of coins can be established during the creation stage and subsequent ownership transfers can be verified.

2.6 Summary

Hash functions map any value or data to a fixed-size bit string. The hash of a value is typically unique and serves as its identifier. Hash functions play a vital role in constructing tamper-proof data structures like blockchains and Merkle trees.

Hashes provide commitments to data, ensuring its integrity and authenticity. Additionally, Merkle trees serve as accumulators and enable proof of membership for individual elements within a committed set.

Digital signatures function similarly to handwritten signatures, providing authentication in communication. In a blockchain system, all exchanged messages are signed to ensure their authenticity.

We previously introduced ledgers as ordered lists of data values. The blockchain and Merkle tree structures are sufficient to create a centralized ledger, with a central authority responsible for writing to the ledger and multiple parties reading from it.

Digital signatures play a crucial role in transitioning from a centralized ledger to a decentralized one.

To achieve a functional decentralized ledger, three important questions need to be addressed:

1. Who can participate in the ledger, and how are participants chosen?
2. How can users append blocks to the ledger, and how can this process be verified in a decentralized manner?
3. Where should a user append their block? In a decentralized ledger, blocks can be attached to any existing block as perceived by the user.

Chapter 3

Proof of Work and Nakamoto Consensus

Still it is note worthy to remember the two keywords for blockchains, decentralization, and trust. What we have discussed so far was that blockchain data structure enables a tamper-evident and tamper-resistant ledger but only a single party has the privilege to write into the ledger. We saw that this data structure has trust built into it. In this lecture we will go through the idea of decentralization and we will see that how bitcoin deals with decentralization via the Nakamoto consensus protocol.

3.1 Decentralized Blockchain & Nakamoto consensus

We saw the idea of signature on the previous lecture. The identity of the signer is so called the public key, in blockchain or bitcoin terminology is simply called address. In a bitcoin wallet, the address simply reports the public key of the wallet. There is also a secret key corresponding to each public key. Giving away secret key would be tantamount to giving away the access to be able to let other people sign for you .

A coin has a unique identity that is linked to its owner. The owner is the person who received the coin in the last transfer transaction, which can be traced back to the original coin creation transaction or the Genesis block.In blockchains, there are special people who have the authority to create new coins. We will discuss who they are, how they got this power, and what are the principles and rules of coin creation in blockchains. This is related to the field of tokenomics, which studies the design and economics of tokens.

A signature can also be on a block itself. If there are several people who are writing to a database, then it may make sense to know who has signed it and who has entered the data. This data structure, or ledger, can be updated by a fixed and known group of parties who may not fully trust each other. They have a common interest in maintaining the ledger, but they need to follow certain rules to add new blocks to it. This could be consortium.

Unlike some blockchains that limit the number of participants, Bitcoin is open and unlimited. Anyone can create as many identities as they want by generating public and private keys. Bitcoin is a free and decentralized system.

3.1.1 Distributed consensus

When different people can update the ledger, who keeps track of all the changes? One way is to have everyone store the whole ledger, but this is very inefficient and costly. We will see how we can relax this assumption and make the system more scalable.They follow a protocol that lets them check and verify each other's blocks. This is how they ensure the ledger's integrity and security. Consensus or agreement is at the heart of trust.

Byzantine fault tolerance (BFT) is a property of a distributed system that allows it to reach a consensus among its components, even if some of them are faulty or malicious. BFT is important for ensuring the reliability and security of a distributed system, especially in scenarios where there is no central authority or trust among the participants. BFT is also relevant for blockchain systems, which are distributed networks that store and process transactions without relying on a central authority. Blockchain systems use consensus protocols to ensure that all nodes in the network agree on the same state of the ledger, despite the presence of faulty or malicious nodes. You could use one of these BFT protocols to come up with agreement among the participants on a block.

Bitcoin consensus protocol is so different from BFT protocols in some ways:

- It's decentralized truly in the sense of permissionless.
- It makes less pessimistic network assumptions.

Now there will be a question that **how do people come to consensus?**
Since in case of bitcoin, one can have several identities, voting can not be an answer like it is for democracy. We need a way to have consensus without voting. This was somehow assumed impossible before Nakamoto.

3.1.2 Network Assumptions

People in a consensus need to communicate with each other, in other words wee need a network among the participants. Bitcoin assumptions for this concept are as follows:

- Any node can broadcast to all nodes into the network and it's a fully connected network.
- Every broadcast message reaches every node albeit with some delay which is about 10 minute for bitcoin.

These concepts and ideas will be discussed more on future lectures.

3.1.3 Leader election: Oracle

One way to quickly decentralize is to elect a leader. The leader's job is to basically put the block together and and put it out for everybody to share it. This sharing the block and creating it is called proposal, and this leader is said to be a proposal. Adding a new block to the ledger is a special role that requires some rules and restrictions. Otherwise, there would be too much conflict and confusion among the nodes. They need to agree on the same ledger state. A proposal is simply identified with a public key and not with IP address or network (node).
The ledger is updated in regular intervals, called rounds. Blocks are not created randomly or too frequently (e.g. every second). For example, in Bitcoin, the protocol specifies that a new block is added every 10 minutes. A node creates a new block by collecting all the new and valid transactions that it has seen on the network, and that are not already in previous blocks. This block is the node's proposal for the next ledger state.

A transaction is valid if it meets two criteria:

1. The sender of the coin must have owned the coin in a previous block in the ledger
2. The sender must not have spent the coin twice in different transactions. This ensures that the coin is not duplicated or forged.

In a glance here are the 4 main activities that a proposal do:

1. constitutes a block with transactions
2. validates transactions
3. includes hash pointer to previous block
4. signs the block

A malicious proposer could try to cheat by signing a block when it was not their turn, or by inserting fake or invalid transactions in the block. However, this can be detected and prevented by the other nodes, who can verify the signature and the transactions. The proposer cannot fool the system by being dishonest or lazy. Everyone can verify the validity of transactions by using the ledger, which is a chain of blocks. Each block has a Merkle root that preserves the order of the transactions in the block. The ledger gives a complete history of all transactions ever made, and allows anyone to track the current state of the coins and their owners. This is how validation is done.

There are two kinds of messages that are broadcasted in the network: transactions and proposals. Transactions are sent by anyone who wants to transfer coins, and they are stored in a temporary memory pool by the nodes. Proposals are created by special nodes who have the right to make new blocks, and they contain some transactions from the memory pool. The proposer can choose which transactions to include in the block, as long as the block size is one megabyte.

A satoshi is the smallest unit of a bitcoin, equivalent to 0.00000001 BTC. There are 100 million satoshi in one bitcoin.

3.2 Proof of Work

the method to elect a proposer that Bitcoin shows here in general is called proof of work. The goal of the competition is to select one of the participants to be the proposer. The competition proceeds like this. Proof of work requires miners to solve complex mathematical problems using their computing power, which consumes a lot of energy. The problems are hard to solve but easy to verify, and the probability of finding a solution is proportional to the amount of work done. The first miner who finds a valid solution gets to create a new block of transactions and receive a reward in Bitcoin. The new block is then broadcasted to the rest of the network and added to the chain of previous blocks, forming the blockchain.

The mathematical problem is actually a hash puzzle. Hash puzzles are a game in which one tries to find a nonce (an integer) such that

$$H(nonce, data) < T$$

where T is the target difficulty level. If you can find a nonce, that's the proof of what is work here. We include nonce inside the block. Threshold is chosen such that a block is mined successfully on average once in 10 minutes. The process of searching for a nonce that solves the hash puzzle is called mining.

Properties of Proof of Work

1. Random miner selected at each time
2. Independent randomness across time and across miners
3. Probability of successful mining proportional to fraction of total hash power
4. Sybil resistance
5. Spam resistance
6. Tamper proof – even by the proposer!

The chance of winning is proportional to how much hash power this miner brings to the competition. Hash power relates to the number of hashes I can try per second, which is directly proportional to how much GPUs energy i can bring.
Sybil resistance is the ability of a system to prevent or deter malicious actors from creating multiple fake identities or nodes to manipulate or attack the system. For example, in a peer-to-peer network, a sybil attacker could create many fake nodes to isolate, censor, or deceive honest nodes, or to gain more influence or voting power.

3.3 Forks and Longest Chain Protocol

Forks and the longest chain rule are related to how the Bitcoin network reaches a consensus on the state of the ledger, which is a chain of blocks that contains all the transactions ever made.

A fork is a situation where there are two or more competing versions of the ledger, each with a different block at the end. A fork can happen when two miners find a valid block at the same time, or when some nodes do not receive or accept a new block. A fork can also be caused by malicious nodes who try to create fake or invalid blocks.

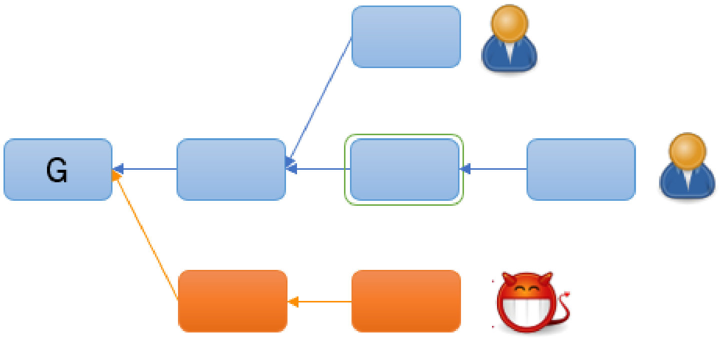


Figure 3.1: Forks in Blockchain

The longest chain rule is a way of resolving forks and choosing the valid version of the ledger. The longest chain rule states that the nodes should always follow and extend the chain of blocks that has the most accumulated proof-of-work, which is a measure of how much energy and effort was spent to create the blocks. The longest chain rule ensures that the majority of the network agrees on the same ledger, and that any forked or orphaned blocks are discarded.

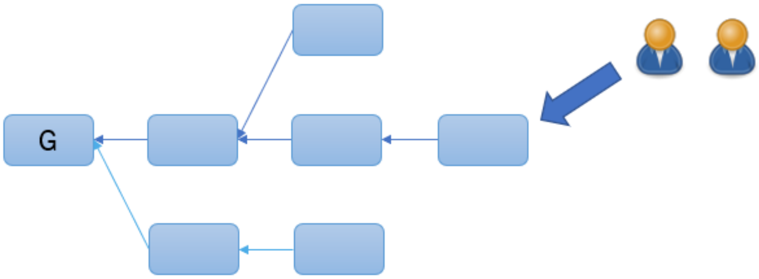


Figure 3.2: The longest chain rule