Principles of Blockchains
Princeton University,
Professor: Pramod Viswanath

Notes - complete edition

August 8, 2023

Contents

Chapter 1 **Introduction** Page 5

1.1 What are blockchains 5
1.2 Bitcoin as the first blockchain 7
1.3 What this course is about 9
1.4 What is Rust? 10

Chapter 2 **Blockchains as Cryptographic Data Structures** Page 11

2.1 Hash Functions 11
2.2 Cryptographic Hash Function 12
2.3 Hash Pointer and chains of block 12
2.4 Merkle tree 13
2.5 Digital Signature 15
2.6 Summary 15

Chapter 3 **Proof of Work and Nakamoto Consensus** Page 17

3.1 Decentralized Blockchain & Nakamoto consensus 17
Distributed consensus — 17 • Network Assumptions — 18 • Leader election: Oracle — 18
3.2 Proof of Work 19
3.3 Forks and Longest Chain Protocol 19
3.4 Adversarial users 20
3.5 The k-deep confrmation rule 22
3.6 Variable mining difculty 24
3.7 Bitcoin is Permissionless 26

Chapter 4 **Peer to Peer Network** Page 28

4.1 Blockchains and Networking 28
Types of Network Architecture — 28 • Overlay Networks — 28 • Gossip and Flooding — 29 • Expander Graph — 29
4.2 Bitcoin Network 30
Peer discovery — 30 • Block transmission — 30 • Data Broadcast — 31 • Compact Blocks — 31 • Disadvantages of current p2p network — 32
4.3 Geometric Random Network 32
Perigee — 33
4.4 Efficient Networking 33
FRN (Fast relay network) — 34 • Trusted Networks: Falcon — 34

4.5 Network Anonymity and Privacy 35
Network De-anonymization Problem — 35 • Flooding Protocol — 36 • Anonymity in Botnet adversarial model — 37 • Dandelion — 38

Chapter 5 Bitcoin system Page 40

5.1 Basic requirements for a banking system 40
5.2 Bitcoin and Satoshi 40
5.3 Transactions 40
Addressing — 41 • Transaction inputs and outputs — 41 • Signatures on transactions — 42 • UTXO — 42 • Cryptocurrency wallets — 43 • Transaction fees — 44 • Coinbase transactions — 44 • Transaction mempool — 45
5.4 Validating a block 47
The state of the system — 47 • Validating blocks — 48 • Light nodes and stateless clients — 49
5.5 Smart contracts 49
Scripts in Bitcoin — 50

Chapter 6 Safety of Bitcoin Page 51

6.1 Introduction 51
6.2 Mining as a Poisson process 51
6.3 Nakamoto consensus protocol model 53
6.4 Formal defnitions of safety 54
6.5 Private attack 54
6.6 Analysis of the private attack under zero delay 55
6.7 Private attack is the worst-case attack 56
6.8 Safety analysis under bounded network delay 57
6.9 Nakamoto blocks 58
6.10 Importance of synchronous network 60

Chapter 7 Liveness of Bitcoin Page 61

7.1 What is Liveness 61
7.2 Chain Growth 62
7.3 Chain Quality 63
7.4 Selfish Mining 64
7.5 Fruitchains 64
Cryptographic Sortition — 64 • Fruitchain Protocol — 65 • Short time scale optimal CQ — 66

Chapter 8 Scaling Throughput in Bitcoin Page 68

8.1 Bitcoin's throughput is security limited 68
8.2 GHOST - embrace forking 69
Private attack on GHOST — 69 • Balance attack on GHOST — 69
8.3 Reduce Forking and Bitcoin-NG 70
Bribing attack in Bitcoin-NG — 71
8.4 Prism 1.0 72

Chapter 9 Scaling Latency Page 73
9.1 Bitcoin Latency 73
9.2 Prism: Fast Confrmation 74
9.3 Fast confrmation rule in Prism 77
Fast confrmation rule — 78 • Tentative Confrmation rule — 79 • Final confrmation rule — 79
9.4 Non-isolated Proposer Block 79
Honest transaction — 80 • Double-spent transaction — 80
9.5 Scaling latency via Prism 80

Chapter 10 Sharding: Scaling Storage, Computation and Communication Page 82
10.1 Introduction 82
10.2 Multiconsensus architecture 83
10.3 Uniconsensus architecture 84
10.4 Bootstrap and State-commitment 86
Accumulator for State-commitments — 87 • Generation and agreement on state commitments — 87
10.5 Cross-shard transactions 89

Chapter 11 Proof of Stake Page 92
11.1 Proof of Work is energy inefficient 92
11.2 Proof of Stake 92
Idea 1 — 93 • Idea 3 — 93
11.3 Key Evolving Signature (KES) 94
11.4 Nothing at Stake attack 95
11.5 Boosting Security Threshold 96
c-correlation PoS protocol — 97
11.6 Dynamic Stake 98
New fork choice rule — 99
11.7 Dynamic Availability Using VDF 100
PoSAT — 100

Chapter 12 Layer 2 Scaling: Side Blockchains Page 102
12.1 Introduction 102
12.2 Side Blockchains 102
Data Availability Attack — 103 • Data Availability Oracle — 103 • Erasure Coding — 104 • Coding integrity and correctness — 105
12.3 AceD 105
Coded Interleaving Tree — 106

Chapter 13 Layer 2 Scaling: Payment Channels Page 109
13.1 Payment Channels 109
13.2 One-Way Payment Channel 110
13.3 Two-Way Payment Channel 111
13.4 Multi-hop payment channels 112
Trusted multi-hop payments — 112 • Trustless multi-hop payments using Hashlock — 112 • Trustless multi-hop payments using Hashlock and Timelock — 112

13.5 Payment Network 113

Chapter 14 Layer 2 Scaling: Rollups Page 115

14.1 Introduction 115
14.2 Transfer 115
14.3 Optimistic Rollup 116
Fraud Detection and Proofs — 116 • Data Availability — 119 • Pros and Cons — 119
14.4 ZK Rollup 119
Validity Proof — 120 • Data availability — 120 • zkSync vs. zkPorter — 121 • EVM Compatibility — 122 • zkEVM — 123 • Pros and Cons — 123
14.5 Final Comparison 123

Chapter 15 Blockchains with Finality Page 125

15.1 Review of the longest chain protocol 125
15.2 Byzantine Fault Tolerant (BFT) Protocols 126
15.3 Streamlet 126
Confirmation rule — 126 • Streamlet performance — 128 • Latency of Streamlet — 129
15.4 HotStuff 129
From Streamlet to Hotstuff — 130
15.5 implementation: Pacemaker 130

Chapter 16 Algorand Page 131

16.1 From Permissioned to Permissionless: Committee Selection 131
16.2 Player Replaceability and Secret Committee Election 132
Ephemeral keys — 134
16.3 Algorand: Single Round BFT Consensus Protocol 134
16.4 Consensus On a Binary Value 135
16.5 Adding Player Replaceability And Secret Committee Election 135
16.6 Multivalue Consensus 136
16.7 Conclusion 136

Chapter 17 Longest Chain Protocol Meets BFT Page 138

17.1 Hybrid Consensus 139
17.2 The CAP Theorem 140
Proof of CAP Theorem — 141
17.3 Finality Gadgets 141
A two-layer design — 142 • Two ledgers and their properties. — 142 • Examining Adaptivity and Finality — 143 • checkpointed longest chain rule — 144

Chapter 18 Data Privacy via Zero Knowledge Cryptography Page 145

18.1 Introduction 145
Trusted third-party mixer — 145
18.2 Zero-knowledge proof 146
Zerocoin — 146 • Zcash — 147

18.3 Zero-knowledge Model 148
Language and NP — 148 • Prover and Zero-knowledge — 148 • Efficiency — 149
18.4 From Zero Knowledge Proof Systems to Zcash 150
18.5 Zcash Framework 150
First Attempt: use commitment — 151 • Second Attempt: two commitments — 151
18.6 Zcash Protocol: Putting it all together 152
Create a pour transaction — 153 • Generate a zk-SNARK proof — 153 • Incentives in Zcash — 154
18.7 Privacy 154

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,

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

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).

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

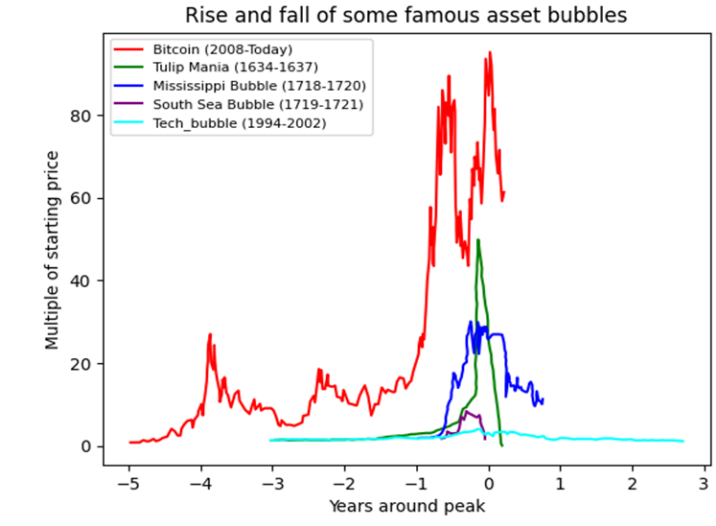


Figure 1.3: Bitcoin and Bubbles

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).

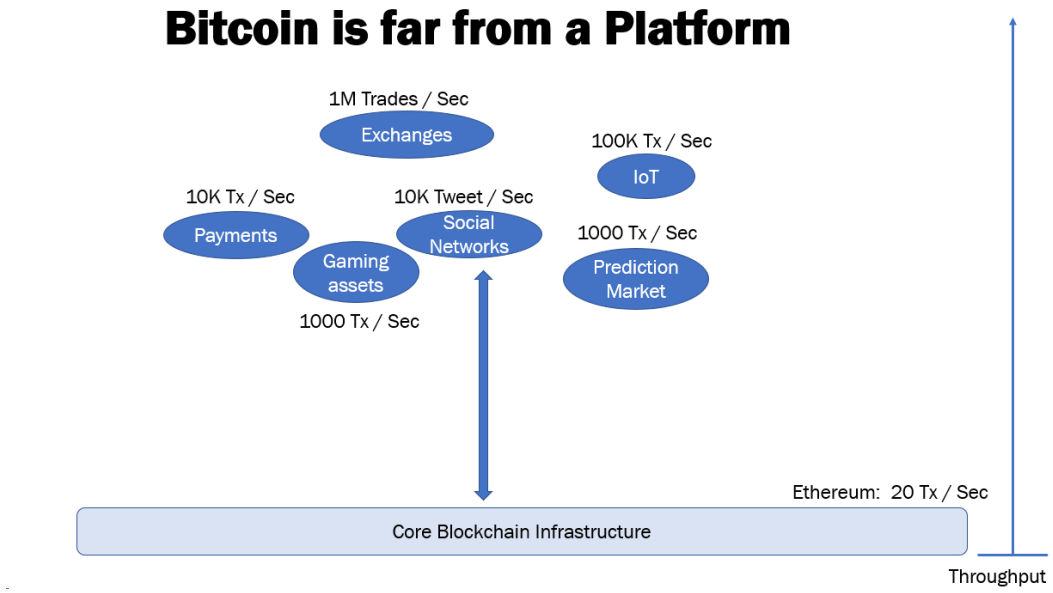


Figure 1.4: Evolution of Trust over human history

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.

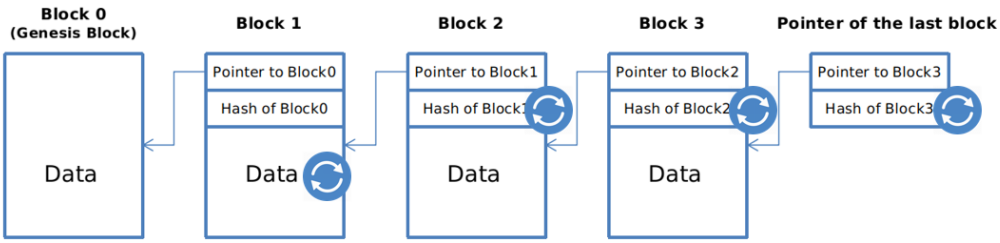


Figure 2.1: A block chain is a linked list that is built with hash pointers instead of pointers

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

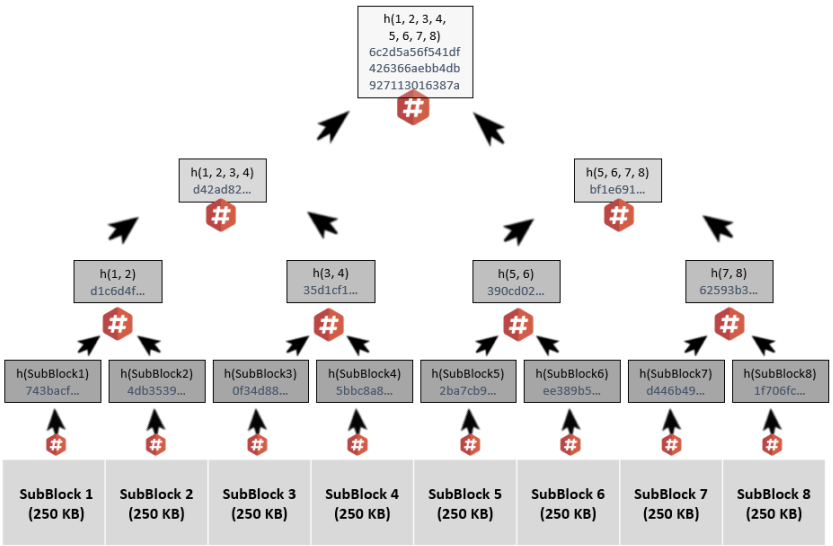


Figure 2.2: A Merkle tree

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.

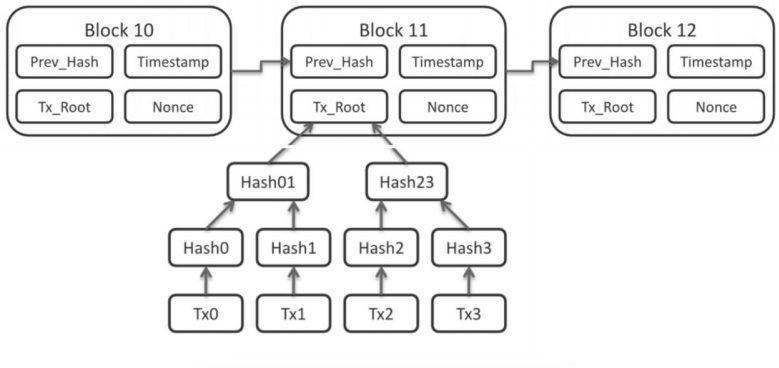


Figure 2.3: Blockchain with markle trees

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.

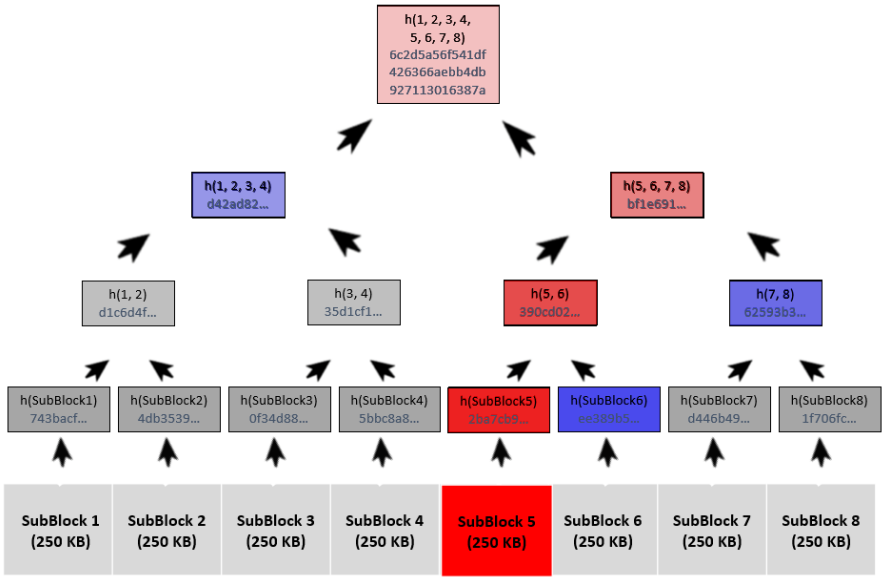


Figure 2.4: Finding a block in Merkle tree

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.

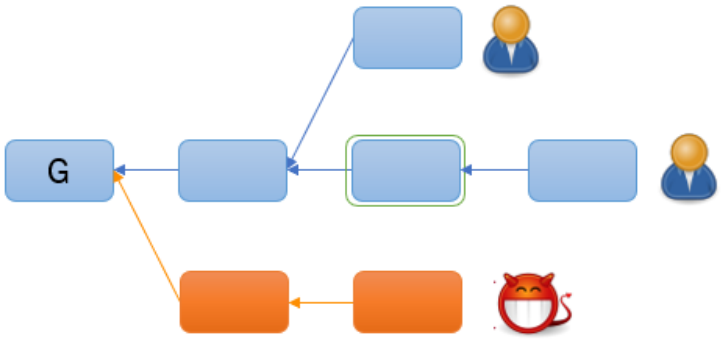


Figure 3.1: Forks in Blockchain

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.

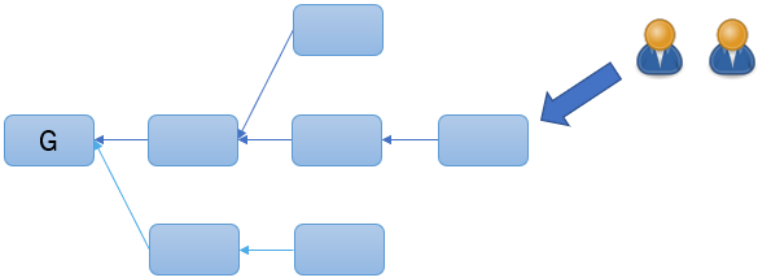


Figure 3.2: The longest chain rule

3.4 Adversarial users

The preceding description outlines the fundamental elements of the Nakamoto consensus protocol, with additional details to be covered in the forthcoming lecture. It is crucial for all users to strictly adhere to the protocol's guidelines. However, in real-world scenarios, there might be some users who deviate from the protocol, and these individuals are known as adversarial users or malicious users.
The primary objective of adversarial parties is to disrupt the system in any way possible. One particular attack they may employ to undermine the append-only property of the ledger is referred to as the **private attack**.
In the private attack scenario, the adversarial users engage in mining new blocks, but they refrain from broadcasting these blocks to the network. This secretive approach allows them to create an alternative chain, or "fork," in the blockchain that remains hidden from honest users. While the adversarial users continue to mine privately, honest users carry on with their mining activities, unaware of the existence of the private blocks.
Due to the inherent randomness in the mining process, the adversarial users might get lucky and quickly build a few private blocks in succession, let's say five blocks. Meanwhile, the honest users, during the same period, mine only three blocks on the public blockchain.

In this situation, the private (adversarial) chain becomes longer than the public (honest) chain because it has five blocks compared to the honest chain's three. Now, the adversarial users release their private chain to all other users, revealing the blocks they kept hidden.
As per the Nakamoto consensus protocol's longest-chain rule, honest users must adopt the chain with the greatest length. Since the private chain is now longer than the public chain, all honest users are compelled to give up their chain and switch to the adversarial chain. As a result, the last three blocks, which were part of the honest chain, are effectively erased from the ledger.
This action of adopting the longer chain effectively invalidates the transactions contained in the last three blocks of the honest chain, allowing the adversarial users to carry out a double-spending attack. By creating a longer chain with more proof-of-work, the adversarial users successfully overwrite the transaction history on the blockchain, causing a disruption and undermining the append-only property of the ledger. This highlights the importance of consensus mechanisms in preventing such attacks and maintaining the integrity and security of decentralized ledgers.

Figure 1.3 illustrates the fork in the blockchain resulting from a private attack:

1. **Scenario:**
   A group of malicious users launches a private attack on the blockchain system.

2. **Fork in the Blockchain:**
   The attack causes a fork in the blockchain, resulting in two competing chains of blocks.

3. **Block Representation:**

   - Dashed blocks: These represent the privately held blocks generated by the adversarial users.
   - Solid blocks: These represent the publicly visible blocks mined by honest users and part of the main blockchain.

4. **Private vs. Public Chain:**

   - Private Chain: Comprises the dashed blocks, which remain hidden from the public network.
   - Public Chain: Consists of the solid blocks and is the main blockchain known to honest users.

5. **Objective:**
   The adversarial users aim to create a longer private chain compared to the public chain by exploiting the longest-chain rule in the Nakamoto consensus protocol.

6. **Revealing the Private Chain:**
   Once the malicious users perceive that their private chain is longer, they reveal it to the entire network.

7. **Switching to the Longer Chain:**
   Following the longest-chain rule, honest users must adopt the longer chain, abandoning the public chain they were initially working on.

8. **Impact on the Ledger:**
   The switch to the adversarial chain effectively invalidates the transactions contained in the blocks of the public chain that are no longer part of the main blockchain.

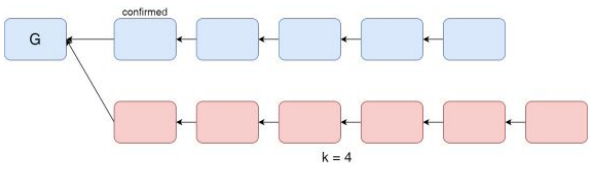


Figure 3.3: An Illustration of the private attack

This private attack undermines the integrity of the blockchain and demonstrates the significance of robust consensus mechanisms in preventing such disruptions and maintaining the security of decentralized ledgers.

The successful execution of private attacks can severely undermine trust in the blockchain system. One particularly impactful demonstration of this is the double-spend action enabled by the private attack. Let's explore this scenario using Figure 1.4:

1. **The Initial Transaction:** A transaction (tx) is embedded in a block B. For instance, let's assume the transaction involves the payment of seven Bitcoins to Tesla, which has recently announced plans to accept Bitcoins for their cars.
2. **Confirmation of the Transaction:** Upon seeing the transaction (tx) embedded in a block on the longest chain, Tesla confirms the transaction and releases the car to the source of the transaction. This action "confirms" or completes the transaction (tx) based on the longest chain, which Tesla considers valid.
3. **Launching the Private Attack:** After the transaction (tx) has been confirmed, an adversary observing this development decides to launch a private attack. The adversary releases two or more blocks it had been secretly mining in private.
4. **Impact of the Private Attack:** As a result of the private attack, the block B containing the confirmed transaction (tx) is no longer part of the longest chain. Consequently, the transaction (tx) is no longer included in the ledger maintained by the network's participants.
5. **Double-Spending Opportunity:** Now that the transaction (tx) has been removed from the ledger, the source of the transaction is free to double-spend the seven Bitcoins. This means they can use the same seven Bitcoins to initiate another transaction, effectively spending them twice.
6. **Trust Demolished:** The successful double-spending attack seriously undermines the trust embodied by the blockchain system. It compromises the integrity of the ledger and erodes confidence in the security of the network.

One way to address this vulnerability is by delaying the confirmation of transactions. By waiting for more blocks to be added to the chain after the transaction is initially embedded, the likelihood of a successful double-spending attack through a private attack is reduced. This delay allows the network to have more confidence in the validity of transactions before confirming them, enhancing the security and trustworthiness of the blockchain system.

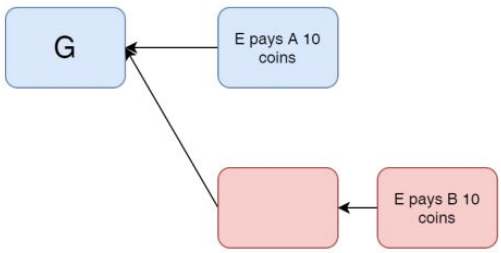


Figure 3.4: An Illustration of the double spend attack.

3.5 The k-deep confrmation rule

In the Nakamoto consensus protocol, occasional changes at the end of the ledger is impossible to avoid. This chancges can be made by adversarial or by network delays.
Malicious users can collaborate and launch private attacks, as discussed earlier. By creating a longer chain in secret and releasing it to the public network, they can rewrite the last block of the ledger. This undermines the integrity of the ledger and poses a significant challenge to maintaining trust in the blockchain system.

Even without adversarial users, forks can occur in the blockchain due to network delays. When multiple blocks are mined simultaneously, they can propagate through the network at different speeds, leading to temporary forks. Honest users may work on different branches of the blockchain, resulting in competing chains.
Forks introduce the possibility of the longest-chain switching. As honest users continue mining and adding blocks to their respective branches, one of the branches might eventually become longer, leading to a shift in the longest-chain rule. When this happens, the blockchain system must adapt to the new longest chain, resulting in a rewriting of the last portion of the ledger.
Indeed, the solution to address the issue of occasional changes at the end of the ledger in the Nakamoto consensus protocol is simple and intuitive. By treating a block as confirmed only if it is buried deep enough under other blocks, the blockchain system can enhance its resilience against forks and ensure the immutability of the ledger. Here's a breakdown of the solution:

1. **Confirmation Depth:**
   Users should consider a block as confirmed only if it is buried deep enough under a certain number of subsequent blocks. Let's denote this number as "k."

2. **Consequences of Confirmation:**
   Confirming a block, say block B, means that the portion of the ledger up to that block is now considered immutable and secure. This implies that all transactions contained in the blocks preceding B are considered final and trustworthy.

3. **Delayed Confirmation:**
   The solution suggests that it is not advisable to confirm very recent blocks immediately. New blocks are vulnerable to being overwritten due to forks in the blockchain. By delaying the confirmation of recent blocks, the system allows more time for the network to converge to a single valid chain.

4. **Protection Against Forks:**
   By requiring a minimum depth of k blocks to confirm a specific block, the system provides protection against forks. Blocks that are not part of the longest chain, and therefore part of competing forks, will not be confirmed until a sufficient number of additional blocks are added to the chain, making it less likely for the confirmation to switch between forks.

5. **Increasing Security Over Time:**
   As more blocks are added to the blockchain and the depth of confirmations increases, the security and finality of transactions improve. Deeper confirmations provide greater confidence that the ledger's history is unchangeable.

This approach aligns with the practical behavior observed in many blockchain systems, where users and applications typically wait for a certain number of block confirmations before considering a transaction as fully settled. The higher the confirmation depth (larger k value), the more secure and reliable the transaction becomes.
By implementing this simple confirmation mechanism, blockchain systems can maintain the integrity of the ledger and safeguard against the potential disruptions caused by forks and private attacks, contributing to a more robust and trustworthy decentralized network.
The k-deep confirmation rule, as described earlier, is based on the belief that any changes in the longest chain will typically occur towards the end of the chain. The prefix of the longest chain at a specific time t, obtained by dropping the last k blocks, will continue to remain a prefix of the longest chain at future times as well. It is essential to note that this belief holds for most practical scenarios, but it is not guaranteed with absolute certainty for any finite value of k.
Certain conditions, such as adversarial users investing in significant computing power, disrupting block communication among honest users, or having an extraordinary stroke of luck, can potentially overturn a deeply buried block. However, as the value of k increases, the likelihood of such events happening diminishes significantly.
Lecture 6 demonstrates that the probability of such events decreases exponentially with k, under the assumption that the honest miners control a majority of the total hash power (i.e., more than 50%). This assumption is critical in the Nakamoto consensus protocol, as it ensures that the honest miners collectively have more computational power than adversarial users, making it more improbable for adversarial users to consistently outpace the honest chain in mining blocks.
In summary, the k-deep confirmation rule, although not entirely foolproof, provides a practical and effective approach to enhance the security and reliability of the blockchain system. By requiring a certain depth of confirmations (larger k value) before considering a block as truly confirmed, the system minimizes the risk of chain

reorganizations due to forks and private attacks. The larger the value of k, the lower the likelihood of such disruptive events happening, reinforcing the blockchain's robustness when honest miners have a majority of the hash power.

3.6 Variable mining difculty

In PoW blockchains, adapting to the vast variation in mining power is essential for maintaining stability and security. One way to achieve this is through a difficulty adjustment algorithm. In the case of Bitcoin, the following three core ideas are used in its difficulty adjustment algorithm:
(a) Varying the Difficulty Target: The difficulty target for block mining is adjusted based on the average inter-block time from the previous epoch, which consists of 2016 blocks. If the average time is greater than the desired inter-block time (e.g., 10 minutes), the difficulty is reduced to make mining easier. Conversely, if the average time is less than the target, the difficulty is increased to make mining harder.
(b) Using the Heaviest Chain Rule: Instead of considering the longest chain, the blockchain system determines the valid chain based on the total sum of block difficulties. The chain with the most accumulated proof-of-work (highest difficulty) is considered the heaviest chain and is used to determine the valid ledger.
(c) Mild Difficulty Adjustment: The difficulty is adjusted only mildly at the end of each epoch, and the adjustment is bounded by a factor of 4. This prevents abrupt and drastic changes in the difficulty, ensuring a smoother transition during different mining periods.
While this algorithm may seem straightforward, simpler approaches can lead to dangerous security vulnerabilities. For example, using only the heaviest chain rule (b) and allowing miners to choose their own difficulty can create significant problems.
An initial analysis might suggest that the heaviest chain rule does not provide an advantage for manipulating difficulty. However, this lack of advantage only holds in expectation, and extremely difficult adversarial blocks can introduce high variance, thwarting confirmation rules that confirm deeply-embedded blocks with non-negligible probabilities proportional to the attacker's mining power[1]..
For instance, suppose honest miners adopt the initial mining difficulty defined in the genesis block, with an expected inter-block time of 10 minutes (using 10 minutes as the unit of time). The difficulty unit is defined as the time it takes to mine a honest chain with k blocks. If the adversarial mining power is half of the honest mining power (or 1/3 of total mining power), the adversary can mine a single block as difficult as k honest blocks within k units of time. The adversarial mining process follows a Poisson point process with rate 1/2k, and the number of adversarial blocks mined in k units of time follows a Poisson distribution Poiss(1/2).
Hence the success probability of this attack would be:

$$P(\text{attack succeeds}) = P(\text{Poiss}(1/2) \geqslant 1) = 1 - e^{-1/2} \approx 39.3\%$$

which is a constant independent of k, therefore any k-deep confrmation rule will fail.
Consider a more detailed difficulty adjustment rule involving only (a) and (b), where the adversary can perform a difficulty-raising attack. The attack exploits the ability to create an epoch with extremely close-together timestamps, causing the difficulty adjustment rule to set the difficulty significantly higher for the next epoch. The attacker then utilizes the high variance in mining to execute the attack. Here's a description of this attack:

1. **Adversarial Timestamps:**
   The adversary can manipulate timestamps in their private blocks, allowing them to create an epoch with timestamps that are extremely close together.

2. **Difficulty Setting:**
   The difficulty adjustment rule (a) calculates the difficulty for the next epoch based on the average inter-block time from the previous epoch. If the timestamps in the adversary's private blocks indicate an extremely fast block generation rate, the difficulty for the next epoch will be set very high.

3. **Difficulty-Raising Attack:**
   Let B be the first block of the second epoch in the adversary's private chain, with difficulty X. The chain difficulty of block B is 2016 + X, as it includes the difficulty of the previous 2016 blocks.

4. **Honest Chain Mining Time:**
   Mining an honest chain with chain difficulty 2016 + X, on average, takes 2016 + X units of time.

5. **Attack Preparation Time:**
   Considering the same adversary, it takes, on average, 4032 units of time for them to complete the first epoch in their private chain.

6. **Attack Execution:**
   To succeed in the attack, the adversary needs to mine block B within X − 2016 units of time, which happens with a probability:

$$P(\text{attack succeeds}) = P(\text{Poiss}\left(\frac{X-2016}{2X}\right) \geqslant 1) = 1 - e^{-(X-2016)/2X} \approx 1 - e^{-1/2} \approx 39.3\%$$

The attack's success probability is approximately 39.3%. If the adversary can accomplish this, they can create a difficulty spike for the next epoch, making it exceedingly difficult for honest miners to keep up with the fast mining rate. This high variance in mining can then be exploited by the attacker, allowing them to perform reorganizations, invalidate deeply-embedded blocks, and potentially double-spend.
This difficulty-raising attack highlights the importance of a careful and sophisticated difficulty adjustment algorithm in PoW blockchains. The Bitcoin difficulty adjustment algorithm takes into account a combination of factors, such as average inter-block time, to prevent such attacks and maintain a stable and secure block generation rate over time. By carefully adjusting the difficulty at regular intervals and avoiding abrupt changes, the system can mitigate the risks associated with extreme variations in mining power and ensure the resilience of the blockchain network.
Correct, if X 2016, the success probability of the difficulty-raising attack remains significant, and it becomes more challenging to rely solely on a k-deep confirmation rule. This complex attack, which is only thwarted by the full protocol employing (a), (b), and (c) together, highlights the importance of the comprehensive Bitcoin difficulty adjustment algorithm. Let's formally describe the Bitcoin difficulty adjustment algorithm:
Consider a chain of v blocks with timestamps $(r_1, \ldots, r_v)$. The Bitcoin difficulty adjustment algorithm uses the following fixed parameters:

- τ (set to 4 in Bitcoin): The factor by which the difficulty is allowed to be adjusted mildly every epoch.
- Φ (set to 2016 in Bitcoin): The length of an epoch in the number of blocks.
- Λ_0 (set to 2 weeks in Bitcoin): The expected duration of an epoch.

The difficulty adjustment algorithm aims to keep the average inter-block time close to the target block time (e.g., 10 minutes for Bitcoin) over the long term.

The algorithm operates as follows:

1. Calculate the Time Difference:

$$D = r_v - r_1$$

2. Calculate the Expected Epoch Duration:

$$\Lambda = \frac{\Lambda_0 \cdot D}{\Phi \cdot \tau}$$

3. Adjust the Difficulty:

$$\text{If } D < \frac{\Lambda}{2}, \text{ increase the difficulty threshold by a factor of } \tau$$
$$\text{If } D > 2\Lambda, \text{ decrease the difficulty threshold by a factor of } \tau$$

4. Keep the Difficulty within Bounds:

   Ensure that the difficulty adjustment is limited to a maximum increase or decrease of a factor of τ

By adjusting the difficulty threshold based on the actual epoch duration and comparing it to the expected epoch duration, the algorithm maintains a relatively stable and predictable inter-block time over the course of a long duration. This prevents abrupt difficulty spikes or drops caused by significant changes in mining power and helps maintain the security and stability of the Bitcoin blockchain.

The full protocol, which employs (a), (b), and (c) together, effectively mitigates various types of attacks and ensures the integrity of the blockchain system, preventing malicious actors from manipulating the difficulty to their advantage.

The target calculation function D : Z∗ → R is defned as

$$D(\epsilon) = T_0$$

where T_0 is the initial target as defined in the genesis block, and Φ', Λ, and T correspond to the last block, duration, and target of the last completed epoch, respectively, i.e., $\Phi' = \Phi \left\lfloor \frac{v}{\Phi} \right\rfloor$, $\Lambda = r_{\Phi'} - r_{\Phi'-\Phi}$, and $T = D(r_1, \ldots, r_{\Phi'-1})$. A full and beautiful analysis of the Bitcoin protocol is provided in [2].

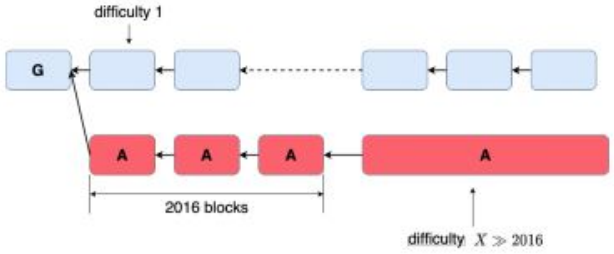


Figure 3.5: The difculty rising attack. The adversary raises the difculty to extremely high in the second epoch by faking timestamps.

3.7 Bitcoin is Permissionless

In the context of Bitcoin, participants have the freedom to generate multiple (secret key, public key) pairs for themselves, allowing them to create multiple identities. This feature is encouraged for privacy reasons, making Bitcoin a permissionless system. However, creating multiple identities does not grant an individual more influence in the system. The actual representation and power in Bitcoin are determined by the computational (mining) power, which can only be increased through a capital investment in hardware and resources. This property ensures that the system is resistant to Sybil attacks, where an adversary could gain an advantage by creating numerous fake identities.

The PoW mining process in Bitcoin serves multiple important purposes simultaneously:

(a) Sybil-Resistance: By tying mining power to computational resources and capital investment, Bitcoin prevents individuals from easily gaining control of the system through the creation of multiple identities.

(b) Randomized Block Proposer Election: PoW mining introduces an element of randomness to the selection of the miner who gets to propose the next block. The probability of being chosen as the block proposer is proportional to the miner's computational power. This process ensures that block proposals are distributed across different miners, enhancing the decentralization of the network.

(c) Adjusting Inter-Block Duration: As discussed earlier, the PoW difficulty adjustment algorithm regulates the inter-block time, maintaining a relatively stable and predictable block generation rate.

In contrast to Bitcoin's permissionless nature, some other blockchain designs implement external mechanisms to ensure that each entity has only a single key, creating a permissioned system. In such permissioned blockchains, separate mechanisms are used for achieving goals (b) and (c) mentioned above, while still maintaining Sybil-resistance through other means.

In subsequent lectures, we will explore some of these other blockchain designs and understand how they address different aspects of consensus, security, and decentralization. Each design may have its trade-offs and may be better suited for specific use cases depending on the requirements of the application.

References

[1] Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013, 2013*

[2] Garay, J., Kiayias, A., & Leonardos, N. (2020). Full analysis of Nakamoto consensus in bounded-delay networks. Cryptology ePrint Archive, Report 2020/277. https://eprint.iacr.org/2020/277

Chapter 4

Peer to Peer Network

4.1 Blockchains and Networking

A blockchain network is a system that provides ledger and smart contract services to applications without relying on a centralized server or authority. A centralized server can be a single point of failure or a target for censorship, which can compromise the security and availability of the network. Therefore, a blockchain network has the following networking requirements:

- The key primitive of the network is to **broadcast blocks and transactions to all nodes**.
  Blocks are data structures that contain transactions, which are records of value transfers or contract executions. Transactions are validated and appended to the ledger by consensus algorithms. Broadcasting ensures that all nodes have the same view of the ledger and can verify its integrity.
- The network also needs to be **robust and resilient to node failures or attacks**.
  Some nodes may go offline due to various reasons, such as power outages, network disruptions, or malicious attacks. The network should be able to tolerate a certain percentage of node failures without affecting its functionality. Moreover, the network should allow new nodes to join and synchronize with the existing nodes, as well as handle node churns and partitions.

4.1.1 Types of Network Architecture

There are two types of network architecture:

1. **Client Server** :
   This is a type of network architecture where one or more servers store most of the data and resources, and the clients access them through requests. The server is the central authority that controls the network, and the clients are dependent on the server for their functionality.
2. **Peer to Peer** :
   This is a type of network architecture where each node acts as both a client and a server, and there is no central authority or hierarchy. The nodes share their data and resources directly with each other, without relying on a server.

The need for robustness implies that we do not want a client-server relationship; we settle for a peer-to-peer (P2P) network where each node has identical behavior.

4.1.2 Overlay Networks

An overlay network depicts the connections between nodes, and is represented as a graph. It abstracts out the physical network switches and routers and defines virtual links between nodes. Two nodes that are connected by a link can exchange messages directly. Those that are not connected by a link must find a path connecting them on this overlay network in order to communicate messages.
There are different ways to classify overlay networks, but one common way is to distinguish between two types of overlay networks

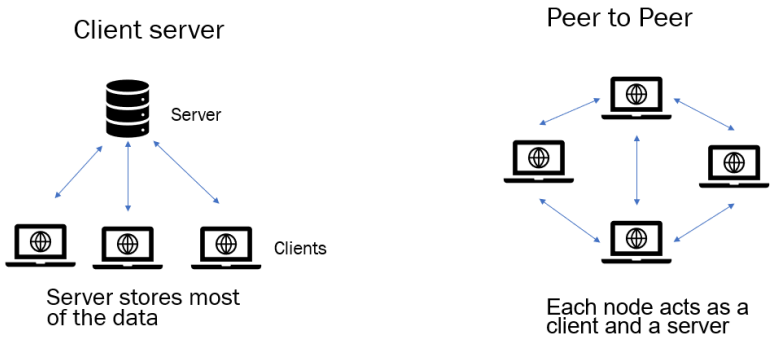


Figure 4.1: Types of Network Architecture

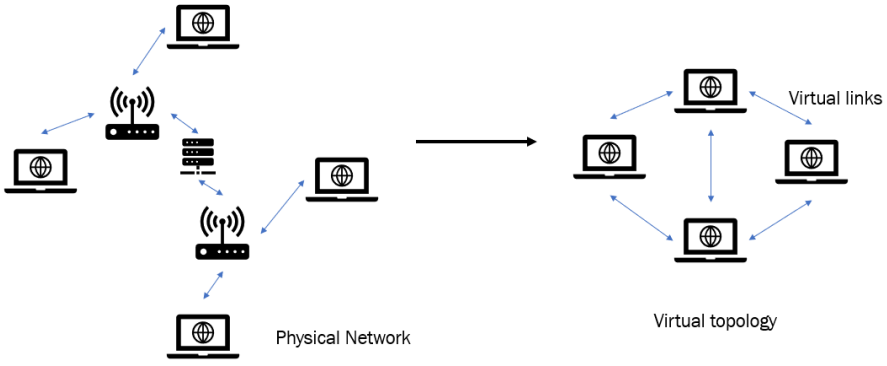


Figure 4.2: Overlay Network

- **structured** : These are overlay networks that have a specific topology or organization, such as a ring, a tree, or a grid. Structured overlay networks, like CHORD, assign an identifier to each node and uses that to construct welldefined routing rules. These networks are excellent for routing sending point to point messages, a use case not required for Bitcoin. Structured networks are suitable for broadcast, too; any message transmission takes O(logN) hops on CHORD with O(logN) connections per node.
- **unstructured** : These are overlay networks that have no specific topology or organization, and the nodes are connected randomly or based on some criteria, such as proximity, interest, or availability. Unstructured networks like d-regular graphs have no node identifiers; a node connects to d other nodes randomly.

Routing point to point messages is not feasible because it takes O(logN) hops and requires many peer queries to find the path from point A to point B. However, broadcast is very effective using gossip and also takes O(logN) hops. Therefore, the number of hops needed is equal to the structured network with the extra advantage of O(1) peer connections. That is why the Bitcoin network uses an unstructured d-regular overlay network.

4.1.3 Gossip and Flooding

Gossip and Flooding are two techniques for disseminating information in a network of nodes. Gossip is a technique that randomly sends the information to some of its neighbors, while Flooding sends the information to all of its neighbors. Both techniques can spread the information exponentially and reach all nodes in O(logN) time.
Gossip and Flooding can be used to achieve fast and reliable information dissemination in a network, but they differ in terms of efficiency, overhead, and robustness.

4.1.4 Expander Graph

Expander graph is a type of graph that has the property of being well connected but sparse.

Definition 4.1.1: Sparse Graph

A sparse graph $G(V, E)$ is a graph that has a small number of edges compared to the number of vertices, such that $|E| = O(|V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices.

An Expander graph is a sparse graph that is also well connected, such that for any subset A of vertices, the number of vertices outside A that have at least one neighbor in A, denoted by $|A|$, is large.

A gossip message starts with $A(0)$ as the broadcasting node with $|A(0)| = 1$. In the next hop, it will reach $\partial A(0)$ with $|A(1)|$ being at least $(1+\epsilon)$ times $|A(0)|$. This process repeats and we get $|A(k)|$ being at least $(1+\epsilon)^k$ times $|A(0)|$ for any k. Therefore, the number of steps to get to half the number of nodes is proportional to the logarithm of the number of nodes. It can be proven that the other half of the nodes can also be reached in O(logN) time.

4.2 Bitcoin Network

Bitcoin is a peer-to-peer network that uses TCP protocol to exchange data. In Bitcoin network:

- The codebase limits the number of outgoing connections to eight and the number of incoming connections to 117.
- The network has a high churn rate (rate at which users join or leave the system); therefore, the node must be prepared to connect to new peers.
- The node maintains a large list of nodes running Bitcoin in the form of their (IP, port) pair and connects to one of them randomly when a slot becomes available.
- The node ensures that the peers it connects to are selected randomly.

4.2.1 Peer discovery

A node starts its list of peers by connecting to a group of DNS seed nodes. These are special nodes that provide a list of IP addresses and ports of active nodes in the network. A node can query a DNS seed node to get this list and then try to connect to some of the nodes in the list.
The seed nodes are not very distributed; so it is not wise to depend entirely on the peer list given by them. After connecting to the first set of peers, a node requests their peer list using **getAddr** and **Addr** messages. The node updates its peer list frequently by swapping peer lists with its peers.

4.2.2 Block transmission

The process of transmitting blocks and transactions involves the following steps:

1. A node sends an **inv** message to its peers, which is an inventory message that tells them what blocks and transactions are available.
2. When a peer receives an **inv** message, it checks if it already has the block or the transaction in its local storage. If not, it sends a **getData** message to the node to get those blocks and transactions.
3. Optionally, the peer can request only the headers of the blocks first, using a **getHeaders** message, before asking for the full blocks .
4. This header-first block transmission can reduce the bandwidth use and prevent invalid blocks from being accepted .

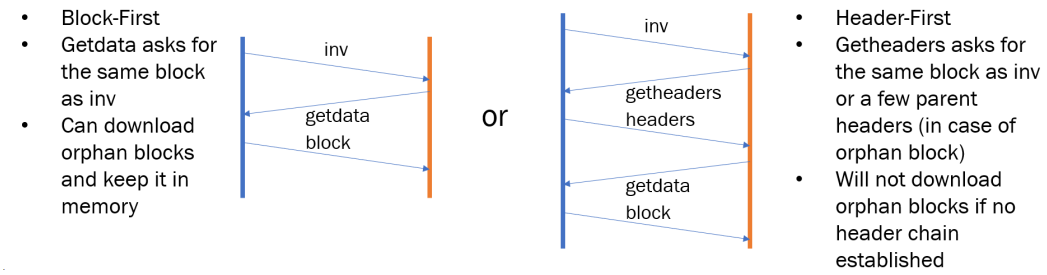


Figure 4.3: Block transmission in blockchain network

4.2.3 Data Broadcast

Now we explain how data, such as transactions, are broadcasted in a peer-to-peer network

- Each node maintains a non-persistent memory to store unconfirmed transactions, which are transactions that have not been included in a block yet. This memory is called the mempool.
- When a node receives a new transaction, it sends an inv message to its peers, which contains the transaction id (txid). The inv message is a way of checking if the peer already has the transaction in its mempool.
- If not, the peer sends a getdata message to request the full transaction data from the node. The node then sends the transaction data (tx) to the peer.
- Some unconfirmed transactions might be removed from the mempool due to various reasons, such as low fees, expiration time, or double spending.

4.2.4 Compact Blocks

Here we discuss two methods of relaying blocks in a peer-to-peer network:

1. **legacy relaying** :
   Legacy relaying is the traditional way of sending the full block data to each node.
2. **compact block relaying** :
   Compact block relaying is a protocol that reduces the amount of bandwidth and latency required to transfer a block that confirms many transactions that the nodes already have in their mempools.

There some differences between the two methods in terms of the messages exchanged between the nodes .Here are the advantages of compact block relaying over legacy relaying:

- Compact block relaying uses shortened transaction identifiers (txids) instead of full transaction data, which reduces the size of the block data.
- Compact block relaying allows nodes to request only the transactions they are missing from their mempools, which reduces the redundancy of sending transactions twice.
- Compact block relaying has two modes: low bandwidth mode and high bandwidth mode.
  Low bandwidth mode minimizes the bandwidth usage by asking for permission before sending a compact block.
  High bandwidth mode reduces the latency by sending a compact block as soon as possible without asking for permission.
- Compact block relaying can verify the proof of work (PoW) from the block header before requesting the full block data, which prevents invalid blocks from being accepted.

There is a trade-off between capacity and propagation delay in a peer-to-peer network. **Capacity** (C) is the maximum amount of data that can be transmitted in a given time, measured in bits per second (bps). **Propagation delay** (D) is the time it takes for a signal to travel from one node to another, measured in seconds. The end to end delay, which is the total time it takes for a block to reach all nodes in the network, increases with the increase

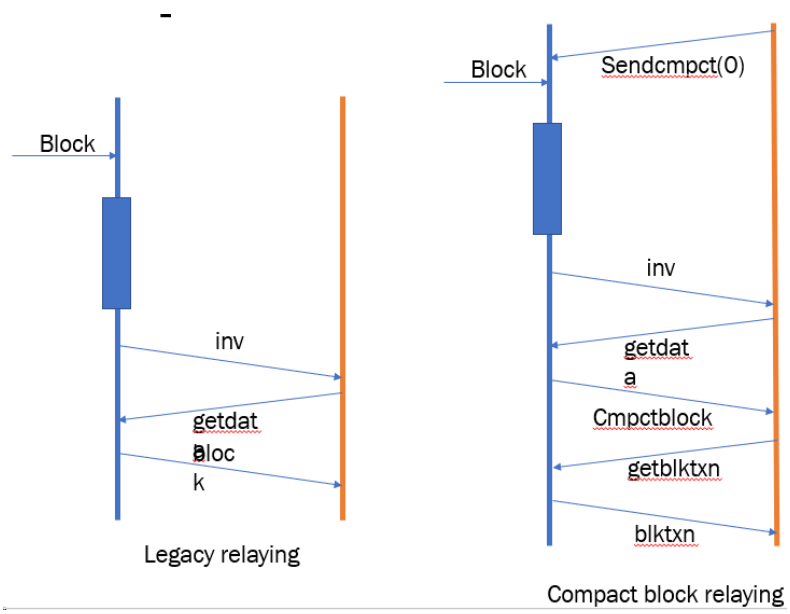


Figure 4.4: Compact Block

in block size. This is because larger blocks take longer to transmit and verify, which can cause congestion and orphaning. Therefore, there is a trade-off between increasing the capacity and reducing the propagation delay of the network. This increase in delay affect the system in following ways:

- Wasted Hash power
- Forking

4.2.5 Disadvantages of current p2p network

- **Efficiency:**
  It requires a lot of communication between nodes. The total communication is proportional to the number of nodes (N) and the average degree of connectivity (d). This means that as the network grows, the communication overhead also increases, which can affect the performance and scalability of the system.
- **Privacy:**
  It can link the transaction source to the IP address of the node that broadcasts it. This means that anyone who monitors the network traffic can potentially identify the sender and receiver of a transaction, which can compromise their anonymity and privacy.
- **Security:**
  It allows for plausible deniability for forking. Forking can cause inconsistency and double spending. Plausible deniability means that a node can claim that it did not receive a valid block from another node, even if it did, and create a fork on purpose. This can be used as an attack strategy to disrupt the consensus and integrity of the system.

4.3 Geometric Random Network

The current network topology is based on random IP addresses, which are not related to geographic distances. This means that nodes may be connected to peers that are far away from them, which can increase the latency and bandwidth usage of the network.

A better network topology would be based on a **geometric random network**, which is a network where nodes are placed according to some geometric criteria, such as proximity or similarity. This way, nodes can connect to peers that are closer or more relevant to them, which can improve the efficiency and performance of the

network. The challenges are creating a self-adapting network topology based on measurements, such as latency, bandwidth, or trust. This means that nodes can dynamically adjust their connections based on the changing conditions and preferences of the network.

4.3.1 Perigee

Perigee is a self-adaptive network topology algorithm for peer-to-peer networks. Perigee is a decentralized algorithm that selects neighbors based on past interactions. It aims to retain neighbors that relay blocks fast and disconnect from neighbors that do not relay blocks fast. It also explores unseen neighbors to discover new potential connections.

'Perigee is motivated by the multi-armed bandit problem, which is a problem of finding the optimal strategy for choosing among several options with uncertain rewards. Perigee tries to balance between exploration and exploitation, which means finding new neighbors and using existing neighbors, respectively. Perigee can improve the efficiency and performance of the peer-to-peer network by reducing the latency and bandwidth usage.

Algorithm

The Perigee Algorithm works as follows:

- It assigns scores for each subset of neighbors based on how fast they relay blocks.
- It retains the subset of neighbors with the best score and disconnects the node that is not in the subset.
- It forms a connection to a random neighbor to explore new potential connections.

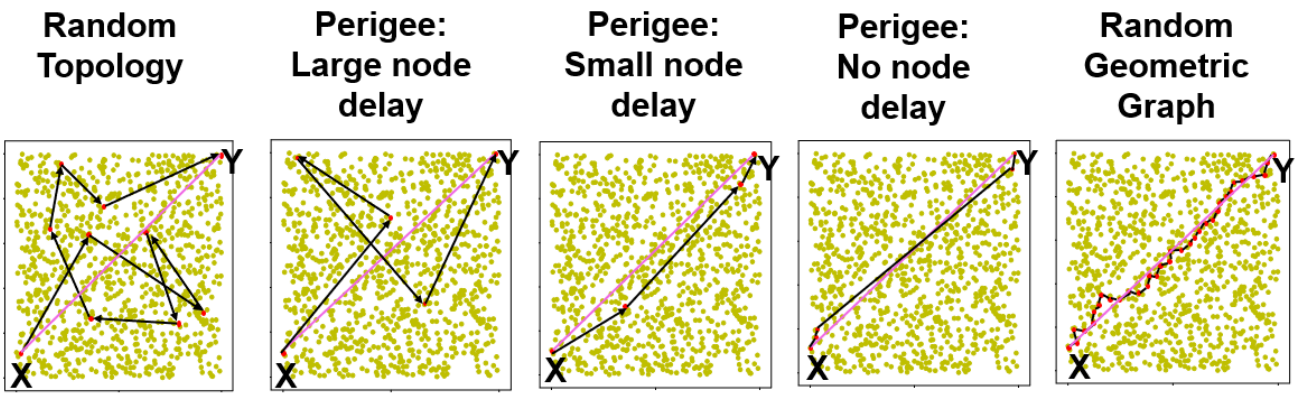


Figure 4.5: Performance of Perigee

4.4 Efficient Networking

Efficient networking is the ability to transmit and receive data in a fast, reliable, and scalable way. There are three aspects of efficient networking:

1. **Trusted networks** :
   Trusted networks are networks where nodes can trust each other to relay valid and timely data. Trusted networks can improve the efficiency of the network by reducing the overhead and redundancy of data transmission. However, trusted networks also pose some risks, such as centralization and censorship.

2. **Privacy** :
   Privacy is the ability to protect the identity and activity of nodes from external observers. Privacy can improve the efficiency of the network by reducing the exposure and vulnerability of nodes. However, privacy

also poses some challenges, such as linking transaction source to IP address and plausible deniability for forking.

3. **Security** :
   Security is the ability to prevent and resist attacks from malicious nodes or adversaries. Security can improve the efficiency of the network by maintaining the integrity and consistency of the data. However, security also poses some difficulties, such as eclipse attacks and plausible deniability for forking.

4.4.1 FRN (Fast relay network)

FRN (Fast relay network) is an example of a trusted network model, which is a hub and spoke model where miners connect to FRN servers. FRN servers are trusted servers that are fast and efficient in relaying blocks and transactions. FRN servers can reduce the latency and bandwidth usage of the network by filtering and compressing the data. However, FRN servers also pose some risks, such as centralization, censorship, and single point of failure.

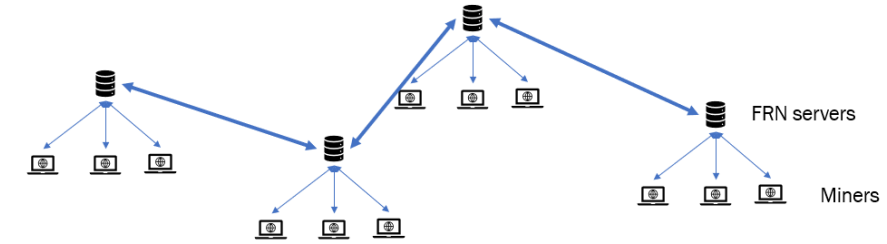


Figure 4.6: FRN

4.4.2 Trusted Networks: Falcon

Falcon network routing is based on the idea of cut through routing, which is a method of forwarding data packets as soon as their headers are verified, without waiting for the whole packet to arrive. This can reduce the latency and bandwidth usage of the network, as well as the risk of forking and double spending. Falcon network routing uses a set of trusted servers, called FRN servers, to relay data packets between nodes.

Cut through routing is the way of sending data packets from point a to point b, where each node verifies the header of the packet and forwards it to the next node without waiting for the whole packet to arrive. The diagram shows that cut through routing has lower latency than legacy routing, which is represented by the distance between the green lines and the red lines in the diagram. The diagram also shows that cut through routing uses FRN servers to relay data packets between nodes.

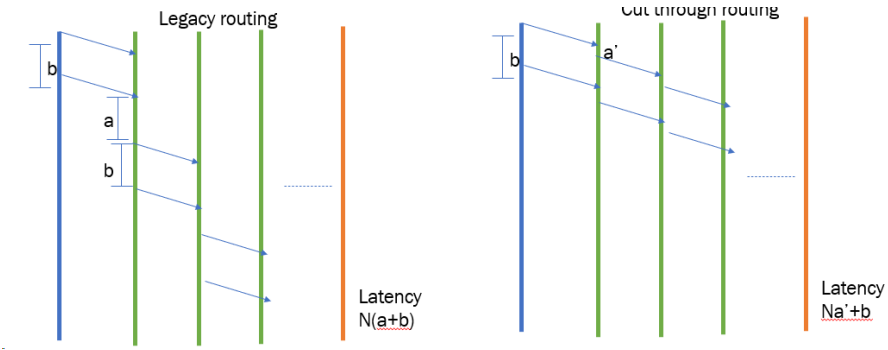


Figure 4.7: Legacy routing vs. Cut through routing

4.5 Network Anonymity and Privacy

Blockchains are systems where the data structure of the blockchain is shared and visible to all participants. For example, in a cryptocurrency, the blockchain records every transaction that ever happened in the currency. However, this also means that the blockchain can contain sensitive data of users, such as their identities and balances. This can raise privacy concerns for users who do not want their data to be public. Therefore, privacy is an important issue in blockchains. How can users achieve privacy in blockchains?

Bitcoin and most other cryptocurrencies use pseudonymous identifiers to protect the privacy of users. Each user has one or more pseudonym, which is a public key that they use to participate in the system. The blockchain only shows the transaction patterns of each pseudonym, not the real identity of the user. However, this system is not very secure, because there are ways to link the pseudonyms to the user's real identity, especially if there is extra information available. These are called de-anonymization attacks, and they can expose the user's sensitive data.

4.5.1 Network De-anonymization Problem

he network de-anonymization problem is the challenge of finding the real identities of users who participate in a network that uses anonymization techniques. Anonymization techniques are methods that hide the personal information of users, such as their names and locations, by using random or fake identifiers. Anonymization techniques are supposed to protect the privacy of users, but they can be broken by attackers who have access to some extra information or techniques. The network de-anonymization problem can be modeled using a graph, where nodes represent users and edges represent connections between them. The attacker's goal is to guess which node in the anonymized graph corresponds to which user in the original graph. The attacker can use different methods, such as eavesdropper adversaries or botnet adversaries, to collect and analyze data from the network. Eavesdropper adversaries are attackers who listen to all or most of the network traffic and use metadata and topology information to infer the source of a message. Botnet adversaries are attackers who control a set of compromised nodes that participate in the network normally and share information with each other. The network de-anonymization problem is a difficult and important problem for network security and privacy.

There are two main types of attackers in network de-anonymization:

1. **Eavesdropper attackers** :
   Eavesdropper attackers are attackers who connect to all or most of the nodes in the network and listen to their communications. They look like normal nodes to the honest nodes, who relay messages to them as usual. Eavesdropper attackers can collect metadata, such as timestamps and IP addresses, from the messages they receive. They can also use information about the network topology, such as the connections between nodes, to guess the source of a message. Eavesdropper attackers usually do not send any messages; they only observe and analyze the messages they receive.

2. **Botnet attackers** :
   Botnet attackers are attackers who control a set of compromised nodes that act like normal nodes in the network. They can accept and relay messages from other nodes, but they also share information with each other. Botnet attackers can have limited visibility into the network, depending on how many nodes they control and how they are connected. Botnet attackers can also inject fake or modified messages into the network to confuse or mislead other nodes.

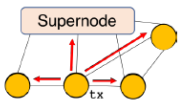


Figure 2: Eavesdropper adversary. A well-connected supernode eavesdrops on relayed communications.

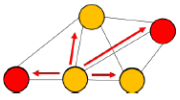


Figure 3: Botnet adversary. Red nodes represent corrupt "spy nodes", which use observed metadata to infer the transaction source.

Figure 4.8: Eavesdropper adversary (Left graph) - Botnet adversary (Right graph)

De-anonymization algorithms are methods that can break the privacy of users who participate in a network that uses anonymization techniques. Anonymization techniques are ways of hiding the personal information of users, such as their names and locations, by using random or fake identifiers. However, these techniques can be defeated by attackers who have access to some extra information or techniques. De-anonymization algorithms are based on the idea of centrality or symmetry, which means that the source of a message is usually located at the center of a disc-shaped region on the network graph. The attackers can use metadata and topology information to infer the shape of the disc and identify the central node with a high probability. These algorithms show that diffusion, which is a common way of spreading messages on the network, is not very good at protecting the anonymity of users at the network level. Therefore, there is a need for alternative spreading protocols that can protect the anonymity of users.

4.5.2 Flooding Protocol

Flooding protocols are methods of sending packets to all nodes in a network by having each node rebroadcast the packets it receives.Flooding protocols can be useful for applications such as firmware updates, routing information, and multicast forwarding. However, flooding protocols can also cause problems such as network congestion, packet loss, and high energy consumption.

There are two flooding protocols:

1. **Trickle** :
   Trickle uses a **polite gossip** policy to control the send rates of nodes. Trickle divides time into intervals and each node maintains a counter of how many consistent packets it has heard from its neighbors. If the counter is below a threshold, the node transmits its packet with a high probability. Otherwise, the node transmits its packet with a low probability. This way, Trickle ensures that nodes hear enough packets to stay consistent, but not too many to cause redundancy.

2. **Diffusion** :
   Diffusion uses an exponential backoff mechanism to reduce the send rates of nodes. Diffusion assigns each node an exponent value based on its hop distance from the source node. The exponent value determines how long the node waits before transmitting its packet. The closer the node is to the source, the smaller the exponent value and the shorter the waiting time. The farther the node is from the source, the larger the exponent value and the longer the waiting time. This way, Diffusion reduces the number of collisions and retransmissions in the network, and improves the reliability and efficiency of packet delivery.

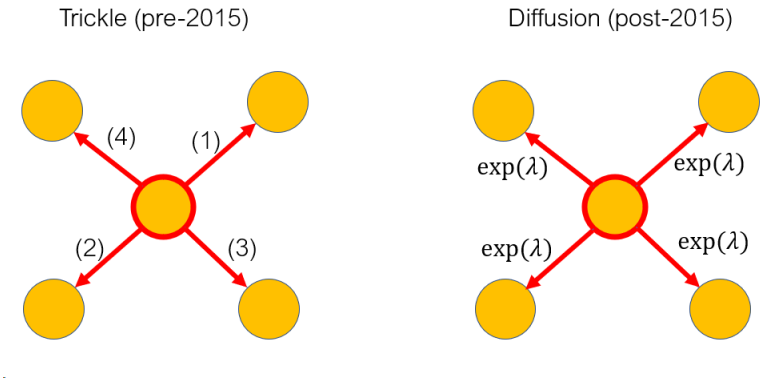


Figure 4.9: Flooding Protocols

Figure 4.10 shows that Trickle has a higher probability of detection than Diffusion for any given number of eavesdropper connections. This means that Trickle is more effective in identifying and reporting eavesdropper connections than Diffusion.
Diffusion does not have (significantly) better anonymity properties than trickle.

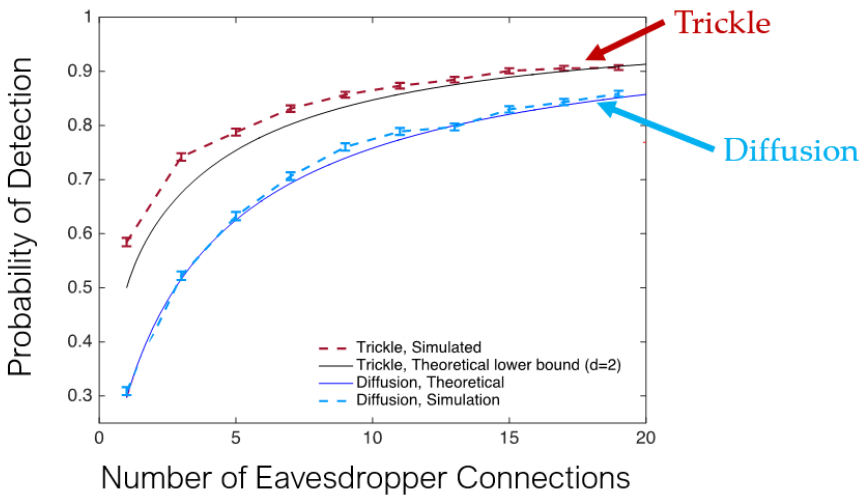


Figure 4.10: Performance of Trickle and Diffusion

4.5.3 Anonymity in Botnet adversarial model

Anonymity is the degree to which a user's identity and transactions are hidden from other users and observers. Anonymity can be measured by two factors: recall and precision. Recall is the probability that a user's transactions are detected by an observer, while precision is the accuracy of the observer's detection. Recall can be calculated

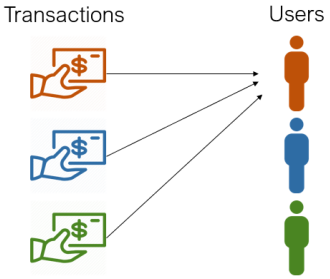


Figure 4.11: Mapping M

using the following formula

$$\frac{1}{n}\sum_{v} 1\{M(v' \, s \, \text{tx}) = v\}$$

where n is the number of honest users in the network, v is a user index, $M(v' \, s \, \text{tx})$ is the mapping function that assigns a user's transactions to a user index, and $1\{M(v' \, s \, \text{tx}) = v\}$ is an indicator function that returns 1 if the mapping is correct and 0 otherwise.

The formula for calculating precision is :

$$\frac{1}{n}\sum_{v} \frac{1\{M(v' \, s \, \text{tx}) = v\}}{\#\text{ tx mapped to } v}$$

Our goal is to design a distributed flooding protocol that minimizes the maximum precision and recall achievable by a computationally-unbounded adversary.

Figure 4.12 implies that we can't have high precision-low recall and vice versa.

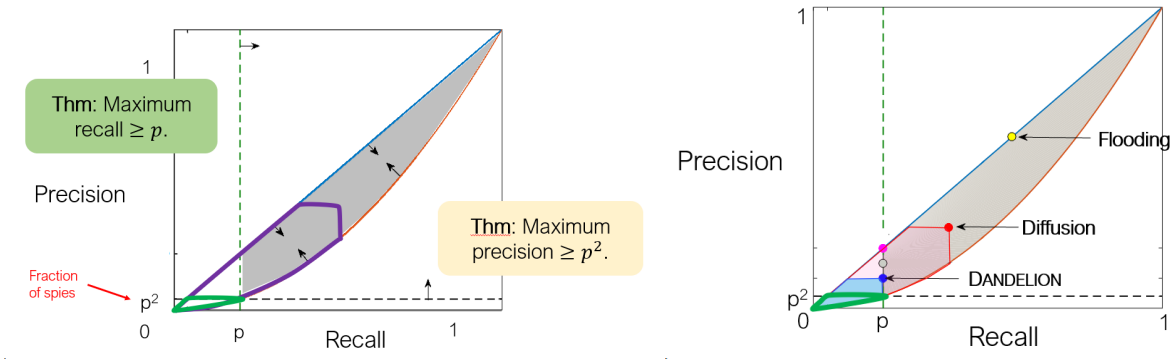


Figure 4.12: Fundamental Limits

4.5.4 Dandelion

One of the main insights from studying how information spreads is that we need to make the communication protocol asymmetric, so that the adversary cannot easily figure out the message. This is the idea behind the Dandelion P2P network protocol, which has two stages:

1. a **stem stage (or anonymity stage)** :
   In the stem stage, each node sends each message along a random direction. This stage lasts for a random number of hops

2. a **fluff stage (or diffusion stage)** :
   in the fluff stage, the message is diffused to the whole network by broadcasting it

The stem stage provides anonymity, while the fluff stage ensures fast and reliable delivery. We will describe the

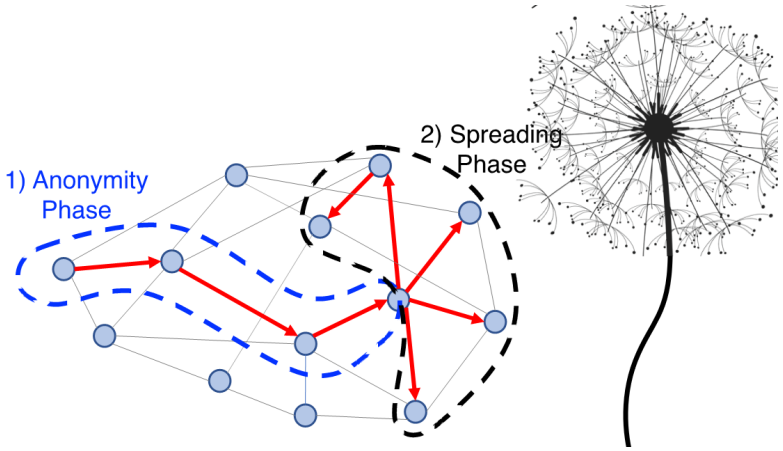


Figure 4.13: Spreading Protocol: Dandelion

Dandelion protocol in more detail next. Dandelion works in epochs that are not synchronized; each node changes its epoch when its internal clock reaches a random value (in practice, this will be a few minutes). Within an epoch, the main algorithmic components of Dandelion are:

1. **Anonymity graph:**
   The random walk happens on a layer of the P2P graph called the anonymity graph. This layer should be either a random cycle graph (i.e., a graph with two edges per node) or a graph with four edges per node. This graph with four edges per node is created from the P2P graph by having each node pick (up to) two of its outgoing edges, without repeating, randomly as Dandelion relays. This does not make an exact graph with four edges per node, but a close one. Each time a node moves to the next epoch, it chooses new Dandelion relays.

2. **Sending of a node's own transactions:**
   Each time a node creates a transaction, it sends the transaction in stem phase along the same random edge on the anonymity graph. If the anonymity graph is a cycle, there is only one edge per node; otherwise, the node must pick one of its two edges.

3. **Relaying of other nodes' transactions** :
   Each time a node gets a transaction in stem phase from another node, it either passes the transaction on or broadcasts it. The decision to broadcast transactions is pseudo-random, and is based on a hash of the node's own identity and epoch number. Note that the choice to broadcast does not depend on the transaction itself—in each epoch, a node is either a broadcaster or a passer for all passed transactions. If the node is not a broadcaster in this epoch (i.e.,it is a passer), then it passes transactions pseudo-randomly; each node assigns each of its incoming edges in the anonymity graph to an outgoing edge in the anonymity graph (with repetition). This assignment is chosen at the start of each epoch, and determines how transactions are passed.

4. **Robustness mechanism**:
   Each node keeps track, for each transaction in stem phase that it sends or passes on, whether the transaction comes back as a transaction in fluff phase within some random time. If not, the node begins to broadcast the transaction.

Dandelion spreading can achieve the lowest possible recall for any given fraction of spies in the network.

Theorem 4.5.1 Recall of Dandelion

Dandelion spreading has an optimally low maximum recall of

$$p + O\left(\frac{1}{n}\right)$$

.

The theorem implies that no algorithm can reduce the accuracy of detection by an adversary below this bound.

Theorem 4.5.2 Precision of Dandelion

Dandelion has a nearly-optimal maximum precision of

$$\frac{2p^2}{1-p}\log\left(\frac{2}{p}\right) + O\left(\frac{1}{n}\right) \quad \text{for } p < \frac{1}{3}$$

Figure 4.14 is a timeline chart that shows the evolution of privacy technologies in cryptocurrencies.

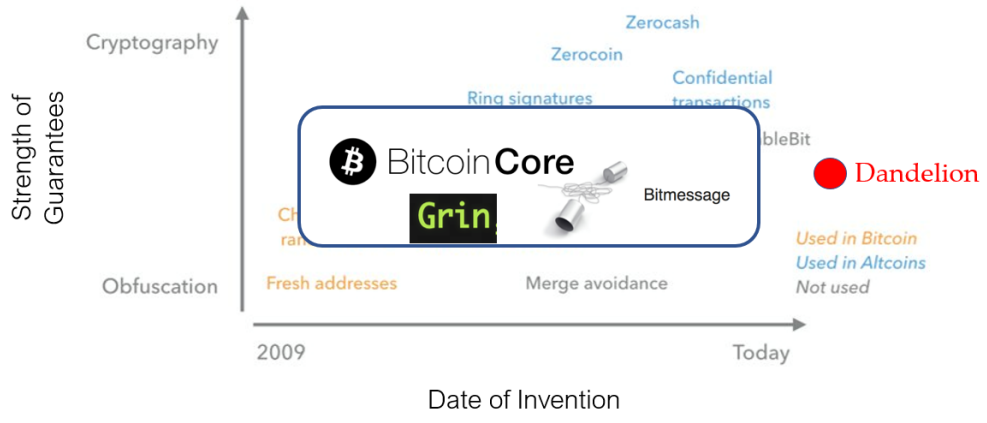


Figure 4.14: Evolution timeline

Chapter 5

Bitcoin system

5.1 Basic requirements for a banking system

In any currency/banking system, there are some basic requirements that the system must provide for. We list them here:

1. There should be a unit of currency/money.
2. There should be a standard way of keeping accounts, i.e., keeping track of how much money each person owns, and transferring money between accounts.
3. No user should be able to create new money from thin air. Put differently, there should be a fixed amount of money in the system at any given time, and new money should be introduced in a systematic manner.
4. A user should not be able to spend more money than he/she owns. There should be a way to verify whether or not this happens.
5. One user should not be able to spend someone else's money (at least, not without their permission).

Let us see how the Bitcoin system provides these features.

5.2 Bitcoin and Satoshi

1. The basic unit of currency in the Bitcoin system is Bitcoin, and the smallest denomination is called a Satoshi, which is equal to 10^{-8} Bitcoins.
2. All transactions in Bitcoin must be an integer multiple of a Satoshi.
3. Just like other currencies, there is an exchange rate between Bitcoin and the dollar. As of July 22, 2023, one Bitcoin is worth 30,000 US dollars. However, the exchange rate between Bitcoin and dollars is very volatile due to various factors, including greed and speculation.
4. The classification of Bitcoin as a currency (like the US Dollar) or a store of value (like gold) has been widely debated. The mainstream view is that Bitcoin is a combination of both.
5. Economically valuing Bitcoin, both in the short and long terms, is an active area of research.
6. Understanding the economic aspects of Bitcoin, both as a currency and a store of value, involves studying various aspects of the Bitcoin system.

5.3 Transactions

In ordinary parlance, the term **transaction** refers to an exchange of something of value. In the context of Bitcoin and cryptocurrencies, a transaction is simply a message that specifes the transfer of money from one entity to another. In fact, transactions are the data-values that get recorded on the blockchain. The blockchain as a ledger is therefore an ordered list of transactions. From this publicly verifable ledger, any user can detect whether transactions are made according to certain rules or not, thereby lending credibility to the ledger and the currency.

5.3.1 Addressing

1. **Bitcoin Address and Its Generation:**
   - In Bitcoin, the notion of traditional accounts is replaced with addresses.
   - An address is simply the hash of a user's public key. It is a unique alphanumeric string that serves as a destination for receiving bitcoins.
   - Addresses are also used to determine where bitcoins will be sent in a transaction.
   - New pairs of public and private keys, and thus new addresses, can be generated at will by a single user.
2. **Receiving vs. Spending Coins:**
   - To receive coins, a user only needs to share their Bitcoin address with others. The address serves as a public identifier for receiving funds.
   - However, to spend coins, a user must also reveal the corresponding public key associated with the address.
   - This is because spending requires providing proof of ownership through a digital signature, which is created using the user's private key.
   - The idiosyncrasy lies in the fact that while an address (hash of public key) is publicly visible and used for receiving, the associated public key is only revealed during the spending process for cryptographic verification.

5.3.2 Transaction inputs and outputs

1. **Transaction Inputs:**
   - Each transaction input represents the amount of Bitcoin being spent from a specific address.
   - When a user initiates a transaction, they reference one or more previous unspent transaction outputs (UTXOs) from the blockchain that they have the right to spend.
   - These UTXOs serve as the inputs to the new transaction and determine the source of the funds being spent.
   - Each input includes a reference to the UTXO's transaction ID and its output index, along with a cryptographic signature to prove ownership.
2. **Transaction Outputs:**
   - Each transaction output represents the amount of Bitcoin being received by a specific address.
   - When a transaction is created, it typically includes multiple outputs, each specifying the amount of Bitcoin and the recipient's address.
   - These outputs determine the destinations of the funds being transferred in the transaction.
   - Each output locks the specified amount of Bitcoin to the recipient's address using a locking script that can only be unlocked with the corresponding private key.
3. **Balance Consistency:**
   - In every valid transaction, the total amount of Bitcoin being spent (sum of inputs) must equal the total amount being received (sum of outputs).
   - This ensures that the transaction preserves the overall balance of the Bitcoin system, i.e., no new bitcoins are created, and no bitcoins are lost during the transaction process.

5.3.3 Signatures on transactions

1. **Signing Transactions for Safety:**
   - Each transaction in Bitcoin must be signed by the users who are spending money. This process is a fundamental safety feature that prevents unauthorized access to someone's funds.
   - For every transaction input, the user creating the transaction must sign it using the corresponding private key associated with the address from which the funds are being spent.
   - By signing the transaction, the user proves ownership of the private key and authorizes the transfer of funds.
2. **One-to-One Correspondence: Public and Private Keys:**
   - As mentioned earlier, each address in Bitcoin has a one-to-one correspondence with a public key, and each public key has a corresponding private key.
   - When a user wishes to spend Bitcoins associated with a particular address, they create the transaction and then sign it using the private key linked to that address.
3. **Verification of Signatures:**
   - After a user signs a transaction, they broadcast it to the network along with the corresponding public key (not the private key).
   - Anyone who sees the signed transaction can verify its authenticity by checking whether it was signed with the private key associated with the public key provided.
   - By doing so, other users can validate that the transaction was indeed signed by the rightful owner of the address (and the coins in that address).
4. **Multiple Addresses, Multiple Signatures:**
   - In transactions that spend Bitcoins from multiple addresses, there must be signatures corresponding to each of these addresses.
   - Each input requires a valid signature to prove ownership and authorization for spending the funds associated with that particular address.

5.3.4 UTXO

The UTXO (Unspent Transaction Output) model is a key feature of the Bitcoin system that ensures the security and validity of transactions. Let's summarize the important points about the UTXO model and how it facilitates transactions with multiple inputs and outputs:

1. **UTXO Model for Validating Transactions:**
   - In the UTXO model, every transaction input must be a transaction output of a previous transaction, linking the spending of funds to their source.
   - When a new transaction output is created, it is considered unspent until it gets consumed as an input in a future transaction when the funds are spent.
   - A valid transaction can only include unspent transaction outputs (UTXOs) as its inputs, providing proof that the address indeed has sufficient funds to spend.
2. **Preventing Double-Spending:**
   - By keeping track of UTXOs at all times, honest users can validate every new transaction against this set to prevent double-spending attempts by dishonest users.
   - To spend their money, users must provide a valid chain of ownership through the UTXO history, ensuring that each input is a previously unspent output.
3. **Multiple Transaction Outputs (Outputs and Change):**

- A transaction may have multiple outputs, allowing users to send funds to multiple recipients in a single transaction.
- For example, if a user owns an address with 2 Bitcoins in an unspent output and wants to spend 1 Bitcoin, the transaction will have two outputs: one for the recipient and one for themselves (the change).
- The change output sends the remaining 1 Bitcoin back to the user's address or a new address.

4. **Multiple Transaction Inputs:**
    - Users can include multiple transaction inputs in a single transaction to combine funds from different unspent outputs for larger transactions.
    - For instance, if a user owns an address with 1 Bitcoin from two separate unspent outputs and wants to pay 2 Bitcoins to another address, they can include both unspent outputs as inputs in the transaction.

The UTXO model's design ensures that each transaction is valid, and its structure allows for flexibility in handling various scenarios, such as spending from multiple sources and sending funds to multiple recipients in one transaction. The adoption of the UTXO model is one of the key factors that contribute to the security and integrity of the Bitcoin blockchain.

5.3.5 Cryptocurrency wallets

Cryptocurrency wallets play a crucial role in simplifying the process of managing and transacting with cryptocurrencies like Bitcoin. Here are the key points about cryptocurrency wallets:

1. **Complexity of UTXO Management:**
    - Keeping track of UTXOs and measuring them for each transaction can be complex and challenging for regular users.
    - Additionally, to maintain anonymity, users often generate new addresses regularly, which requires careful handling of private keys and secure storage.
2. **The Role of Cryptocurrency Wallets:**
    - Cryptocurrency wallets are software applications that handle many of the complex tasks in the background, making it easier for users to transact in cryptocurrencies.
    - Wallets manage UTXOs, track balances, create new addresses, and handle cryptographic signatures for transaction verification.
3. **Anonymity and Security:**
    - Wallets ensure that new addresses (and the corresponding keys) are generated securely without revealing private keys.
    - For maintaining anonymity, wallets often offer features like generating new addresses for each transaction, making it harder to link transactions to a specific user.
4. **Secure Storage and Usage:**
    - Wallets provide a secure storage solution for private keys, protecting them from unauthorized access or theft.
    - Users must be cautious and responsible for securing their wallet's private keys since compromising them could result in the loss of funds.
5. **Trust in Wallet Software:**
    - Using a cryptocurrency wallet requires placing trust in the software's functionality and security.
    - Using a cryptocurrency wallet requires placing trust in the software's functionality and security.

5.3.6 Transaction fees

Transaction fees play a crucial role in the Bitcoin network and serve as an incentive for miners to include transactions in the blocks they mine. Here are the key points about transaction fees:

1. **Total Value in Inputs and Outputs:**
   - In a Bitcoin transaction, the total value in the inputs (UTXOs being spent) must be equal to or greater than the total value in the outputs (newly created UTXOs).
   - The difference between the total value in inputs and outputs is known as the transaction fees.
2. **Transaction Fees as Miner Incentives:**
   - The transaction fees are claimed by the miner who successfully includes the transaction in a block and adds that block to the blockchain.
   - Miners compete to add transactions to their blocks because the fees they collect are an additional reward on top of the block reward (newly minted Bitcoins).
3. **Transaction Prioritization and Speed:**
   - Transactions with higher fees are more attractive to miners because they earn more rewards for including them in their blocks.
   - Miners prioritize transactions with higher fees, leading to faster inclusion of these transactions in the blockchain.
   - Transactions with lower fees may take longer to be confirmed since they are lower in the priority list.
4. **Automatic Fee Calculation:**
   - Wallets automatically calculate transaction fees based on a particular fee rate, measured in Satoshi per kilobyte (Satoshi/kB).
   - The fee rate determines the amount of Satoshi to be paid per kilobyte of transaction data.
   - Higher fee rates result in faster confirmation times, while lower rates may lead to delayed confirmations.
5. **Dynamic Fee Variation:**
   - Transaction fees can vary over time due to fluctuations in network demand and block space availability.
   - During periods of high network congestion, transaction fees may increase as users compete for limited block space.

5.3.7 Coinbase transactions

Coinbase transactions are a crucial mechanism for introducing new Bitcoins into the Bitcoin system and rewarding miners for their efforts in processing transactions and adding blocks to the blockchain. Here are the key points about coinbase transactions:

1. **Introduction of New Bitcoins:**
   - New Bitcoins are introduced into the system as a reward for miners who successfully mine a new block.
   - Every block includes a special transaction known as the "coinbase transaction," which allows the miner to claim a fixed number of newly created Bitcoins.
2. **Block Reward Halving:**
   - Initially, when Bitcoin was launched, the block reward for miners was 50 BTC per block.
   - Approximately every 210,000 blocks (about four years), the block reward is halved through an event known as "halving."
   - This means that after each halving, miners receive half the number of Bitcoins as the previous reward.
3. **Current Block Reward:**

- As of now, there have been three halvings, and the current block reward for miners is 6.25 BTC per block.
- The block rewards will continue until the year 2140 when the total supply of Bitcoin will reach its cap.

4. **Fixed Supply of Bitcoin:**
   - The total supply of Bitcoin is capped at 21 million coins.
   - This fixed supply ensures that there will never be more than 21 million Bitcoins in circulation.
   - Approximately 19.4 million coins are already in circulation, and the remaining Bitcoins will be gradually introduced through coinbase transactions until the cap is reached.
5. **Incentive Mechanism:**
   - Coinbase transactions, along with transaction fees, serve as incentives for miners to actively participate in the network and secure the blockchain through the proof-of-work process.
   - In the early years of Bitcoin, coinbase transactions formed a significant portion of the rewards for miners, but over time, the contribution of transaction fees has increased.

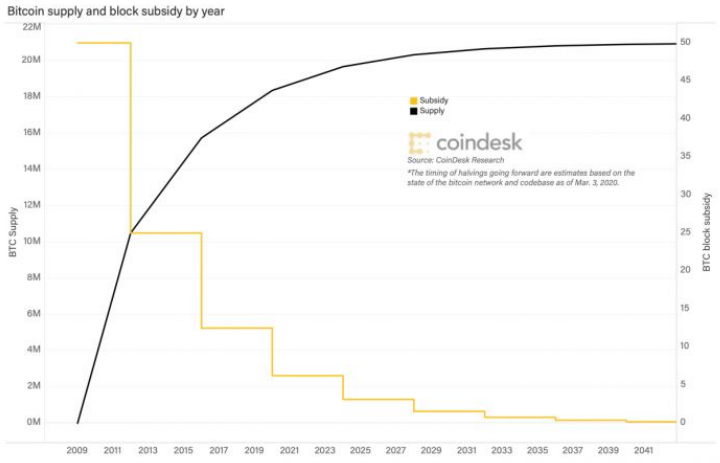


Figure 5.1: Bitcoin block reward in bitcoins. Figure sourced from here

The combination of coinbase transactions and transaction fees ensures the proper functioning and security of the Bitcoin network. Miners are motivated to continue mining, securing the network, and processing transactions as they receive both block rewards and transaction fees as incentives for their efforts. The gradual reduction in block rewards through halvings also creates a deflationary monetary policy, which further contributes to Bitcoin's scarcity and potential value appreciation over time (Figure 1.1, 1.2).

5.3.8 Transaction mempool

The transaction mempool plays a crucial role in the process of how Bitcoin transactions are propagated, verified, and eventually included in blocks. Here are the key points about the transaction mempool and its significance:

1. **Transaction Propagation:**
   - When a user wants to make a Bitcoin transaction, they create a transaction message, sign it using their private key, and broadcast it across the Bitcoin network.
   - The transaction message is a few kilobytes in size and contains information about the transaction inputs, outputs, and the corresponding digital signatures.
   - Miners in the network receive these transactions and verify the signatures before adding them to their memory pool, known as the "mempool."

Figure 5.2: Bitcoin rewards in dollars. Figure sourced from here

2. **The Mempool:**
    - The mempool is a temporary storage area where miners keep pending transactions that they have received and verified.
    - Transactions in the mempool are waiting to be included in a block and added to the blockchain.
    - Miners prioritize which transactions to include in their working block based on factors like the transaction fee rate (higher fee with smaller size).
3. **Block Size Limit:**
    - Each block in the Bitcoin blockchain has a maximum size limit, currently set at 1 megabyte (MB) in the Bitcoin network.
    - Miners aim to maximize the total transaction rewards in their block while adhering to this size limit.
4. **Transaction Fee Rates:**
    - Miners prioritize transactions with higher transaction fees because they want to maximize their potential reward for including transactions in a block.
    - Transactions with higher fees are more likely to be included in blocks sooner, incentivizing users to offer higher fees if they want their transactions processed quickly.
5. **Confirmation Latency:**
    - There is a certain latency between the time a transaction is issued and when it is confirmed on the blockchain. This latency is influenced by two factors: (a) the time it takes for a transaction to be included in a block (can be reduced by offering a higher transaction fee), and (b) the time it takes for that block to be buried deep enough in the longest chain for the transaction to be considered confirmed.
    - Users may choose to trade off latency for security, deciding whether to use higher transaction fees for faster confirmation or lower fees for potentially longer confirmation times.
6. **Blockchain Design Trade-offs:**
    - The trade-off between latency and security is specific to the Bitcoin blockchain design.
    - Other blockchain designs may have different approaches and trade-offs, which will be explored in future lectures.

The mempool and the prioritization of transactions based on fees are essential components that allow the Bitcoin network to efficiently process and confirm transactions while incentivizing miners to maintain the security of the blockchain through the proof-of-work process (Figure 1.3).

5.4 Validating a block

5.4.1 The state of the system

The state of the blockchain system, also known as the UTXO set (Unspent Transaction Output set), is a critical component in ensuring the integrity of the cryptocurrency network. Here are the key points about the state of the system and its significance in Bitcoin:

1. **Verifying Transaction Double-Spending:**
   - One of the essential tasks in the Bitcoin system is to prevent double-spending, where a user attempts to spend the same coin or UTXO more than once.
   - To verify the validity of a new transaction, users need to check if the transaction's inputs (UTXOs) have not been spent previously.
2. **Unspent Transaction Outputs (UTXOs):**
   - The state of the system is represented by the set of UTXOs, which includes all transaction outputs that have not been spent in the blockchain.
   - UTXOs are created when transactions are included in the blockchain and are considered valid until they are spent by new transactions.
3. **Separate from the Mempool:**
   - The state of the system is distinct from the mempool, which contains pending transactions yet to be confirmed.
   - The state consists of UTXOs that have been confirmed and permanently recorded in the blockchain.
4. **Role in Transaction Validation:**
   - When miners construct new blocks from transactions in the mempool, they only include valid transactions by checking the information in the state (UTXO set).
   - Transactions that attempt to spend already spent UTXOs are rejected as invalid.
5. **Summary of Ledger Entries:**
   - The state contains a summary of all the confirmed ledger entries in the blockchain until the current point in time.
   - This summary is crucial for validating new transactions and ensuring their consistency with previous transactions.
6. **Growth and Size Challenges:**
   - The UTXO set grows over time as more transactions are confirmed and new UTXOs are created.
   - As of now, the UTXO set size is quite large, over 5 GB, and it continues to increase as Bitcoin gains popularity.
   - The size of the UTXO set presents scalability challenges, especially concerning memory requirements and specialized hardware for storage and management.

Overall, the state of the system (UTXO set) plays a vital role in maintaining the integrity of the Bitcoin network, preventing double-spending, and ensuring that new transactions are consistent with previous ones. Managing the growth of the UTXO set is an important aspect of addressing scalability concerns in the blockchain system.

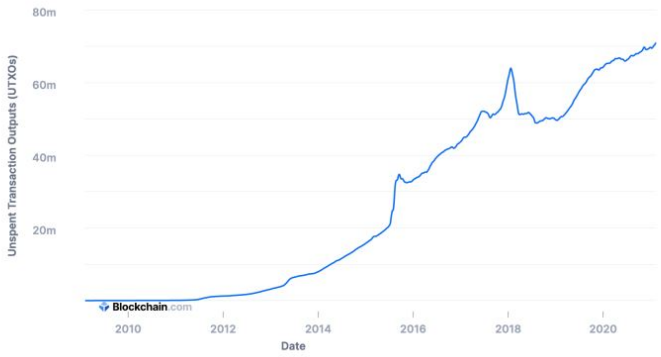


Figure 5.3: Number of UTXOs over time in Bitcoin. Figure sourced from here

5.4.2 Validating blocks

Validating blocks is a crucial step for miners in the Bitcoin network to ensure that the newly received or constructed block adheres to the rules and integrity of the blockchain. Here are the important points about block validation:

1. **Proof-of-Work (PoW) Check:**
   - The first check performed by a miner is to verify whether the proof-of-work presented in the block meets the required threshold.
   - The proof-of-work is a computationally intensive puzzle that miners must solve to create a new block, and it ensures the security and immutability of the blockchain.
   - If the proof-of-work is valid, the miner can proceed with further checks.
2. **Transaction Consistency Check:**
   - After validating the proof-of-work, the miner needs to verify whether the transactions included in the block are consistent with the current state (UTXO set).
   - This consistency check is crucial to prevent any attempts at double-spending or invalid transactions.
   - The miner must confirm that each input in every transaction is a valid unspent transaction output (UTXO) according to the current state.
3. **Updating the State:**
   - If all the transactions in the block pass the consistency check, the miner accepts the block, removes the relevant transactions from its mempool, and updates the state (UTXO set).
   - Updating the state involves deleting the spent transaction outputs (UTXOs) that were used as inputs in the block's transactions.
   - Additionally, the newly generated unspent transaction outputs (UTXOs) are added to the state, representing the funds transferred in the block.
4. **Preemptive Validation in Block Construction:**
   - Before a miner starts mining a new block, it performs the same validation steps preemptively on the transactions it intends to include in the block.
   - This ensures that the constructed block will be accepted by the network once it is mined, saving computational effort.

In summary, block validation is a critical process to maintain the integrity of the blockchain. Miners verify the proof-of-work and the consistency of transactions with the current state (UTXO set). Once validated, a new block is added to the blockchain, and the state is updated to reflect the changes made by the included transactions.

5.4.3 Light nodes and stateless clients

Light nodes and stateless clients are important concepts in blockchain networks, especially for enhancing scalability and allowing users with limited resources to participate in the system. Here's an explanation of these concepts:

1. **Light Nodes / Light Clients:**
   - Light nodes, also known as light clients, are participants in the blockchain network who do not store the full state of the system.
   - Instead, they only verify the proof-of-work on blocks to ensure their validity, and they may selectively verify specific transactions that concern them.
   - By avoiding the storage of the entire state, light nodes require much less computation and storage power compared to full nodes, making them more accessible to devices with limited resources.
2. **Stateless Clients:**
   - Stateless clients are an extension of light nodes that aim to validate transactions and blocks without maintaining the full state of the blockchain at all times.
   - Instead of storing the entire state, stateless clients download only the necessary state information on-demand to validate a specific block or transaction.
   - Once the validation is complete, the stateless client can delete the downloaded state information to save storage space.
   - To securely handle the state information, it is stored in the form of an accumulator, a data structure that we discussed in Lecture 2 of the blockchain course.
3. **Advantages of Stateless Clients:**
   - The adoption of stateless clients can significantly enhance the scalability of blockchain networks. It allows more users to participate without the need for extensive resources to store the full state.
   - By reducing the storage requirements for clients, the network becomes more inclusive and accessible to a wider range of devices, including mobile phones and low-powered computers.
   - Stateless clients can still contribute to the network's security by validating transactions and blocks, albeit without maintaining a persistent copy of the state.
4. **Implementation in Ethereum:**
   - The concept of stateless clients is actively pursued by the Ethereum blockchain network as a scalability solution.
   - Ethereum aims to transition from a stateful design (where clients store the full state) to a stateless design, using stateless clients to improve the network's performance and reduce resource requirements.

In conclusion, light nodes and stateless clients are innovative approaches to make blockchain networks more scalable and inclusive. By allowing participants to interact with the network without storing the full state, these concepts address the challenges of resource-intensive storage and computation in traditional blockchain architectures.

5.5 Smart contracts

In the context of blockchain technology, smart contracts are self-executing contracts with the terms and conditions written directly into code. They are programmed to automatically execute and enforce the terms of the contract without the need for a trusted third party. Smart contracts run on blockchain networks, and their execution is validated and recorded on the blockchain, making them transparent and immutable.
In traditional settings, when two parties want to exchange money subject to certain conditions, they would create a written contract and rely on a trusted intermediary or authority to enforce the contract's terms. With smart contracts on a blockchain, this intermediary is eliminated, and the trust is decentralized across the network.
In summary, smart contracts are a powerful innovation that leverages blockchain technology to enable secure, automated, and transparent execution of agreements without the need for intermediaries. While Bitcoin's smart contract capabilities are limited, other blockchain platforms like Ethereum have expanded the possibilities of smart contracts to revolutionize various industries and applications.

5.5.1 Scripts in Bitcoin

In the context of Bitcoin, smart contracts are referred to as "scripts." A script is a sequence of opcodes included in every transaction output, specifying the conditions that must be satisfied to spend the coins from that output. By using different combinations of opcodes, Bitcoin scripts can enforce various conditions beyond the default ones, providing a level of programmability.

The default condition in a regular transaction output requires the spender to provide a public key that matches the address mentioned in the output and sign the transaction with the corresponding private key. This ensures that only the rightful owner of the coins can spend them.

However, Bitcoin scripts allow for additional conditions to be specified. Some examples include:

1. **Timelocks:** A transaction can be configured to be unspendable until a certain block height or a specific time is reached. This can be useful for time-sensitive transactions or implementing time-based conditions.

2. **Multisig:** A script can require multiple signatures from different private keys to spend the coins. This allows for escrow-like arrangements, where funds are held until all parties involved in the transaction provide their signatures.

3. **Hashed Timelock Contracts (HTLCs):** HTLCs are a specific type of smart contract that combines time locks and hash locks. They are useful for setting up secure escrow arrangements for exchanging assets or payments between parties, ensuring that both parties fulfill their obligations before the funds are released.

The combination of these conditions in Bitcoin scripts enables the creation of more sophisticated smart contracts, even though Bitcoin's scripting language is not as powerful as the one used in platforms like Ethereum. While Bitcoin's script language is intentionally limited for security reasons, it still provides enough flexibility to implement useful smart contract functionalities, especially for simple financial transactions and basic programmability.

Chapter 6

Safety of Bitcoin

6.1 Introduction

The security of a blockchain system, such as Bitcoin, is of utmost importance to ensure the integrity and trustworthiness of the network. When analyzing the security of a blockchain, it is crucial to consider the presence of adversaries who may try to disrupt the system for their own gain. In the context of blockchain security, adversaries are often modeled as powerful entities with significant resources, including more information, computational power, and control over the network compared to honest participants.
Bitcoin incorporates several security measures to protect honest users from adversarial attacks. These measures include:
Proof-of-Work Verification: Honest nodes verify the proof-of-work in new blocks, ensuring that miners have expended sufficient computational effort to create a valid block. This prevents adversaries from easily generating new blocks and taking control of the blockchain.
Transaction Verification: Honest nodes validate signed transactions, ensuring that only rightful owners can spend their funds. Transactions that do not follow the rules are ignored.
UTXO Validation: Honest nodes keep track of the unspent transaction outputs (UTXOs) and verify that each transaction's inputs reference valid UTXOs. Invalid transactions are not accepted.
While these sanity checks are straightforward and easy for honest participants to perform, attacks at the level of the Nakamoto consensus protocol, which determines the longest chain and the agreed-upon version of the blockchain, are more complex and subtle. Adversaries can attempt to disrupt the consensus protocol in ways that are not immediately evident to honest players.
Formal mathematical models are used to analyze the security of the Nakamoto consensus protocol. These models treat the mining process as a stochastic process, where new blocks are generated at random intervals of time. The use of stochastic modeling allows for probabilistic analysis of the protocol's security guarantees. The security guarantees in this context are probabilistic because they depend on the randomness inherent in the mining process.
By applying formal mathematical methods and probabilistic analysis, researchers can gain insights into the security properties of the Nakamoto consensus protocol. This includes understanding the likelihood of successful attacks by adversaries and the resilience of the blockchain system to such attacks.

6.2 Mining as a Poisson process

A Poisson process is a stochastic process where events occur at random intervals, following the exponential distribution. In this case, the events correspond to the mining of new blocks. The intervals between any two block arrivals are statistically identical and independent of each other.
If we denote the time between consecutive block arrivals as X, then X follows an exponential distribution with parameter . The probability that the time between two block arrivals is greater than or equal to t is given by:

$$P(X \geqslant t) = e^{-\lambda t} \quad \text{for all } t \geqslant 0.$$

This probability is often used to analyze the behavior of the Poisson process and to estimate the time it takes for the next block to be mined, given the current rate . The exponential distribution is particularly useful

in modeling Poisson processes, as it captures the randomness and memorylessness properties of the arrival times in the process.
Indeed, a Poisson random variable Y with parameter has a probability mass function given by:

$$P(Y = k) = \frac{e^{-\lambda}\lambda^{k}}{k!} \quad \text{for all } k \geqslant 0.$$

In a Poisson process, the number of events (block arrivals in this case) in an interval of length T follows a Poisson distribution with parameter T, assuming the average rate remains constant. Furthermore, the number of events in disjoint intervals of time are statistically independent.
If we consider small intervals of time (T 1), the probability of having one event (block mined) in the interval is approximately T, and the probability of having no events (no block mined) is approximately 1 - T. This approximation is justified because T is very small, given the average rate is generally small when expressed in terms of the number of blocks per unit of time.
This is where the analogy with coin tosses comes in. A Poisson process can be emulated by counting the occurrence of "successes" (in this case, block arrivals) in a sequence of independent Bernoulli trials (coin tosses), where the probability of success (head) is very small (approximately equal to T).
In modeling the mining process as a Poisson process, we assume that the total number of hashes computed (or nonces tried out) per unit time remains roughly constant, even if the number of miners and their individual computation power may vary. This assumption allows us to treat the mining process as a random process, where new blocks are found at random intervals.
Let's recap some important modeling assumptions about the mining process in the context of the Nakamoto consensus protocol:

1. **Constant Mining Rate ( ):** We assume that the mining rate (the average number of blocks mined per unit time) is a constant, denoted by . In Bitcoin, the average block time is about 10 minutes, so is approximately 1 block every 600 seconds. In Ethereum, the block time is much faster, so the value of is higher.

2. **Poisson Process:** The mining process is modeled as a Poisson process, where new blocks are found at random intervals following an exponential distribution. The Poisson process assumption holds because the probability of successfully mining a block in a given second is very small (0.03 in the example given), and the mining process is essentially like flipping a biased coin (trying out nonces) repeatedly, where each flip is independent of the others.

3. **Independent and Constant Difficulty:** We assume that the difficulty level of the hash puzzle for proposing a block remains constant over some period of time. The difficulty level determines the number of leading zeros required in the hash output, and a high difficulty makes finding a valid block more challenging. In reality, the difficulty is adjusted regularly to maintain a constant mining rate, and this ensures that the time to mine a block remains relatively constant over time.

4. **Uniform Distribution of Valid Nonces:** Valid nonces (hash inputs) are assumed to be uniformly distributed among all possible nonces. This is a common assumption in modeling proof-of-work systems like Bitcoin.

5. **Idealized Network and Synchronization:** The model assumes an idealized network where all nodes receive blocks and transactions immediately, and there are no network delays or partitions. In practice, real-world networks introduce latency and possible partitions, affecting block propagation and communication.

6. **Honest Majority:** We assume that the majority of miners in the network are honest, following the protocol correctly. This is a common assumption in security analysis, as most blockchain systems rely on the assumption that a majority of computational power is controlled by honest participants.

By making some of these modeling assumptions and giving the adversary some extra power, the security analysis can provide guarantees that hold even in practical scenarios where real-world considerations might deviate from the idealized assumptions. The goal is to design a protocol that remains secure and resilient in the face of adversarial attacks, even when the adversary has considerable computational power and control over the network.

6.3 Nakamoto consensus protocol model

In the model, we assume the presence of a single adversary with a fraction of the total computing power (hash power) denoted by β. The remaining computing power, $1-\beta$, is distributed roughly equally among the many honest parties participating in the protocol.
To be more precise:

- The adversary's computing power is β times the total computing power of all users in the system.
- The computing power of honest users is divided equally among them, with each honest user controlling a very small fraction $\frac{1-\beta}{N}$, where N is the total number of honest users.

This means that the adversary mines blocks at a rate of (Poisson process of rate), while honest users mine blocks at a combined rate of $(1-\beta)\lambda$ (Poisson process of rate $(1-\beta)\lambda$). The processes of the adversary and the honest users are independent of each other.
This assumption gives the adversary a significant advantage in terms of mining power, as it controls a substantial portion of the total computing power. The adversary's higher mining rate ($\beta\lambda$) compared to that of honest users ($(1-\beta)\lambda$) allows them to have a higher probability of finding new blocks and building their own chain. However, we still assume that $\beta < \frac{1}{2}$, which means that the honest users collectively have more mining power than the adversary. This assumption is crucial to ensure that the Nakamoto consensus protocol remains secure even in the presence of an adversary.
In the model, we assume that each honest player (party) stores a single blockchain at all times, which is the longest chain they have heard until that specific time. Let's denote the chain held by party h at time i as C_i^h.
Here's a more detailed explanation of the notation:

- i R+: This represents a continuous-time parameter, which denotes the specific point in time during the protocol execution. The time parameter i can take any non-negative real value.
- C_i^h: This notation represents the chain held by party h at time i. The chain C_i^h is a sequence of blocks, starting from the genesis block (the first block) up to the last block at time i that party h has seen and believes to be part of the longest chain.

In this model, honest players continuously update their chain to be the longest chain they have seen so far. This implies that if a party hears of a new block that extends the current longest chain, they update their chain accordingly to include this new block. As a result, the chain C_i^h held by each honest party h may vary over time as new blocks are discovered and added to the blockchain.
The longest chain rule states that honest players always work on extending the longest chain they know, assuming it is the valid chain. As more blocks are mined and added to the blockchain, the chain held by each honest party h (C_i^h) keeps growing as they update it to follow the longest chain.
In the model, when an honest party hears of multiple chains with the same (maximum) length, they choose one of them arbitrarily as C_i^h. This choice is made by the adversary, which means that an honest user may swap its chain when it hears of an equally long chain, not just a strictly longer one. Additionally, if two honest parties both hear of two chains of equal (maximum) length, they may adopt different chains due to this arbitrary tie-breaking power given to the adversary.
To define the notion of the prefix of a chain, we say that chain C1 is a prefix of chain C2 if all blocks in C1 are also present in C2. Formally, we denote this by C1 C2. To represent the prefix of a chain, we use the notation C^h, which represents the prefix chain obtained by dropping the last k blocks from chain C. If chain C has less than or equal to k blocks, C k will be considered as the genesis block.
The k-deep confirmation rule in Bitcoin implies that a node treats all but the last k blocks in its longest chain as confirmed. In the notation of the model, the blocks in C^h are confirmed by user h at time i. Once we confirm a block, we also confirm all the transactions in it. The confirmation rule is applied locally by each user, which means that a transaction confirmed by one user may not be confirmed by another. However, it is desirable that a transaction confirmed by one user becomes confirmed by all other users relatively quickly and remains confirmed forever after. This desirable property is called a safety property in blockchains.
In the context of safety, it is important to ensure that once a block is confirmed (included in the longest chain) by a significant number of honest parties, it is highly unlikely to be reversed or replaced by another chain. In other words, a confirmed block's position in the blockchain should be secure and unlikely to be changed. This property is crucial for maintaining the consistency and integrity of the blockchain, especially in the presence of adversarial

behavior. Ensuring safety is an essential aspect of blockchain protocols to maintain the trust and reliability of the system.

6.4 Formal defnitions of safety

In the Nakamoto consensus protocol, safety is a crucial property that ensures the consistency and integrity of the blockchain. We define safety in two different ways: the common prefix property and individual block safety.
Definition 1 (Common Prefix Property):
The k-common prefix property holds during an execution of the protocol if any block that is committed by one honest user appears in every honest user's chain thereafter. In mathematical terms, for all pairs of times i_1 i_2 and for all pairs of honest users h_1, h_2: $C^k C_2$, where $C_1 C_{i_1}^{h^1} and C_1 C_{i_2}^{h^2}$.
Definition 2 (Individual Block Safety): In an execution, a block B present in some honest user's chain is safe if, after it has been committed by any honest user, it remains a part of all honest users' chains. In mathematical terms, if block B is committed by some user h at time t, then for all tt and for all honest users h, $BC_{t'}^h$.
Note that the common prefix property subsumes individual block safety. For simplicity, the focus is on individual block safety alone.
These definitions represent desired security properties of the protocol. To ascertain whether such properties hold or with what probability they hold, further calculations and analysis are required. Security guarantees or theorems are typically given with the form that the k-common prefix property holds with a probability approaching one as k approaches infinity. These guarantees are made under specific assumptions, including a fixed mining rate, bounds on the adversary's computing power, and bounds on network delays. Importantly, these guarantees are provided while assuming that the adversary can act arbitrarily, meaning no specific assumptions are made about the adversary's strategy.
To understand how the adversary can disrupt safety, it is essential to consider possible adversarial actions. In particular, the focus is on an adversary attempting to disrupt the individual safety of a specific block B. By understanding the various strategies the adversary can employ, a comprehensive analysis of the protocol's security can be performed.

6.5 Private attack

In the simple model of the blockchain system with one adversarial user and many honest users, the adversary aims to violate the safety of a specific block B by performing a private attack. The adversary's goal is to first confirm block B by some (or all) honest users and then dislodge it from the longest chain in the future.
One possible attack is the private attack, previously introduced in Lecture 3. Here's a more detailed description of the attack:

1. The adversary identifies block B's parent, denoted as B, and immediately mines a conflicting block on B right after B is mined. The adversary continues mining privately, extending this hidden chain.

2. Meanwhile, the honest users, unaware of the private chain, continue mining on the longest chain below block B.

3. The honest and adversarial chains are independent of each other and follow Poisson processes with rates $(1\beta)\lambda$ and $\beta\lambda$, respectively. (Recall that represents the fraction of hash power controlled by the adversary.)

To successfully perform the private attack and violate the safety of block B, the adversary must reveal its private chain at the right time. If the adversary reveals the private chain while it is still shorter than the longest honest chain, the honest users will simply ignore the adversary's chain, and the attack will have no effect.
Therefore, the adversary must wait until its private chain is at least as long as the honest chain containing block B (see Figure 1). This way, when the adversary reveals its chain, it will create a fork from a block preceding B and eventually produce a chain of equal or longer length than the longest chain containing B.
Timing is critical for the success of the private attack. If the adversary reveals its chain too early, even if it displaces block B, the attack will not lead to a safety violation. The adversary must wait until the honest chain is long enough, at least k deep, for the attack to have the desired impact and violate the safety of block B.

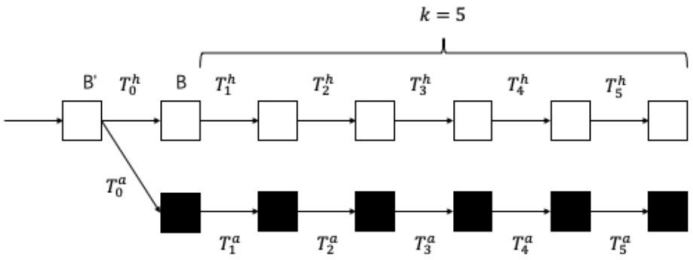


Figure 6.1: Private attack on block B with k = 5.

6.6 Analysis of the private attack under zero delay

Suppose now that the network delay $\delta = 0$. So any node mined successfully by an honest node reaches all other nodes instantaneously. One immediate effect of this is that the honest nodes always have access to the latest mined block and thus can always mine on the tip of the longest public blockchain (by potentially switching their mining upon receiving a new block). Denote the time intervals between the honest and adversarial blocks in the chains of length $k+1$, following block B by $(T_0^h, T_1^h, ...T_k^h)$ and $(T_0^a, T_1^a, ...T_k^a)$ respectively (see Figure 1, where $k = 5$). Then the private attack is successful exactly when the total time to mine $k+1$ blocks by the adversary is faster than the time the honest miners take to mine their $k+1$ blocks, i.e.,

$$\sum_{i=0}^{k} T_i^h \geqslant \sum_{i=0}^{k} T_i^a$$

Note that the times are all independent of each other and $T_0h, ..., T_kh$ are identically distributed as exponential random variables with mean $(\frac{1}{1-\beta\lambda})$ and $T_0^a, ..., T_k^a$ are identically distributed as exponential random variables with mean $\frac{1}{\beta\lambda}$. By the law of large numbers

$$\frac{1}{k+1}\sum_{i=0}^{k} T_i^h - T_i^a - > \frac{1}{(1-\beta)\lambda} - \frac{1}{\beta\lambda}.$$

The limiting value is positive (i.e., the private attack is successful) exactly when

$$\frac{1}{(1-\beta)\lambda} > \frac{1}{\beta\lambda}.$$

or simply $\beta > \frac{1}{2}$, i.e., the adversary controls a majority of the hash power. This calculation shows that in the limit of k very large, the safety of any block B in the longest chain of Bitcoin will continue to remain in the longest chain with probability approaching one. In general there is a tradeoff between a confirmation depth of k and the resulting safety of a block B that has k blocks mined under it. The probability of "deconfirmation", i.e., the block B gets dislodged from the longest chain can be calculated as follows (here $s > 0$ is a parameter to be chosen later):

$$P(\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(s\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)} > 1) \tag{6.1}$$

$$\leqslant E[e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)}] = E[\prod_{i=0}^{k} e^{s(T_i^h - T_i^a)}] = \prod_{i=0}^{k} E[e^{s(T_0^h - T_0^a)}] \tag{6.2}$$

$$= (\frac{\beta\lambda}{\beta\lambda + s})^{k+1}(\frac{(1-\beta)\lambda}{(1-\beta)\lambda - s})^{k+1} \tag{6.3}$$

because of the factorization of the moment generating function of independent random variables. Choosing $s = (1\frac{2\beta\lambda}{2})$ minimizes the exponent and we see that the probability of deconfirmation of a block B is upper bounded

by $e^{c(k+1)}$ where the exponent is given by: $c = \log_e(4\beta(1-\beta)) > 0$. So the probability of deconfrmation decays exponentially in k, as also illustrated in Figure 2. Turns out this calculation was also conducted by Nakamoto themself, albeit somewhat incorrectly, which is included in Figure 2 as a comparison.

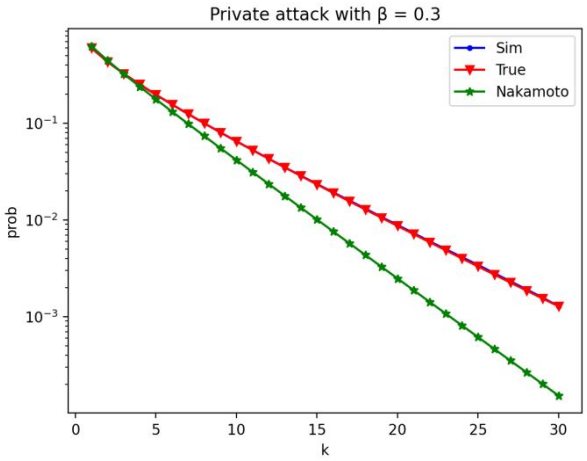


Figure 6.2: Private attack with $= 0.3$.

6.7 Private attack is the worst-case attack

The balance attack is another type of attack on the safety of the blockchain, where the adversary mines and releases blocks in a way that balances the heights of two chains. This prevents any one chain from stabilizing, causing constant switching between the two chains and preventing the ledger from stabilizing.
In the case of zero network delay (where all miners instantaneously learn of any newly mined block), it has been observed that whenever any attack on safety is successful, the private attack is also successful. In this context, safety refers to the k-deep confirmation rule, which means that blocks deeper than k blocks in the longest chain are considered confirmed. Here are some key observations related to this:

1. To launch a safety attack (a balancing attack), at least two chains of length $k+1$ must be present: one chain to confirm a block B (of depth k) and another chain to de-confirm block B by switching the longest chain.

2. Therefore, the total number of blocks mined by both the adversary (A_k) and the honest nodes (H_k) during the attack must be at least $2k+2 : A_k + H_k 2k+2$.

3. Due to zero network delay, two honest nodes can never mine at the same level of the blockchain. Honest nodes always mine on the tip of the longest chain, preventing parallel mining at the same level.

4. For two chains to have a length of at least $k+1$ (to launch the safety attack), the number of adversarially mined blocks must be larger than the number of honest mined blocks: $A_k H_k$.

From these observations, it follows that $A_k k+1$, which means that the adversary has successfully mined at least $k+1$ blocks by the time of the safety attack on the k-deep confirmation rule.
The significance of this result is that, in the case of zero network delay, any attack that successfully violates safety (such as a balance attack) requires the adversary to have already mined at least $k+1$ blocks. And with this number of blocks, the adversary could have used them to launch a private attack instead.
This observation is crucial in understanding the relationship between different attack strategies and provides insights into the security of the Nakamoto consensus protocol under various conditions. It shows that the private attack is a powerful and general attack strategy, and other attacks, like the balance attack, are limited in their effectiveness compared to the private attack (Figure 3).

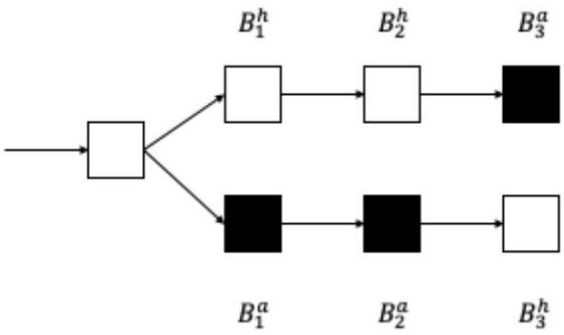


Figure 6.3: Illustration of a balance attack.

6.8 Safety analysis under bounded network delay

In the δ-synchronous network model with propagation delay, there is a finite amount of time δ within which all blocks reach all participants definitively. The adversary can deliver blocks to different participants at a time of its choosing, as long as the δ-synchrony condition is met. This introduces natural forking in the blockchain, even without adversarial intervention.

In the context of Bitcoin, with an inter-block arrival time of ten minutes, natural forking is not very prevalent because it is unlikely that honest nodes will mine on stale blocks. However, in other blockchains like Ethereum, where the inter-block arrival time is fourteen seconds, natural forking is more common due to the shorter block time.

The term $(1-\beta)\lambda\delta$ represents the expected number of blocks that are being mined "in parallel" by the honest nodes. Here, $(1-\beta)\lambda$ is the rate at which honest blocks are mined, and δ is the network delay. This means that during the time λ, the honest nodes may mine multiple blocks, but only one of these blocks will succeed in extending the longest chain. The other blocks mined in parallel are considered "wasted" because they will not become part of the final longest chain.

For example, if the honest fraction of hash power $(1-\beta)\lambda$ is 0.9 and the network delay δ is 10 seconds, then the expected number of parallel blocks mined by honest nodes during this time is 0.9 blocks. Out of these 0.9 blocks, only one block will eventually be included in the longest chain, and the remaining $0.9-1=-0.1$ block is considered "wasted" mining effort.

In blockchains with shorter block times and higher natural forking rates, a larger fraction of honest mining power may be "wasted" due to natural forking. This natural forking phenomenon is an inherent characteristic of the Nakamoto consensus protocol in blockchain systems and affects the efficiency of the mining process.

Thus the rate of growth of the honest chain is now reduced by a fraction $\frac{1}{1+(1-\beta)\lambda\delta}$ to

$$\frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\beta}$$

Now an adversary launching a private attack is successful if its rate of growth (of a private chain) is larger than that of the (reduced) growth rate of the honest chain:

$$\beta\lambda > \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\beta} \tag{6.4}$$

Figure 4 illustrates the minimum hash power the adversary needs to succeed in its private attack, i.e., to replace a certain number of blocks on the longest chain with its own longer chain. The adversary can execute the private attack by mining blocks in secret and then revealing them at a strategic moment to create a longer chain that overtakes the honest chain.

Previously, we saw that when the network delay (δ) is zero, the private attack is a worst-case attack in a very general setting. This means that for every sample path of the mining process and for every confirmation depth (k), the private attack is successful whenever any attack is successful. However, in cases where the network delay is non-zero ($\delta > 0$), the private attack may not always be the worst-case attack.

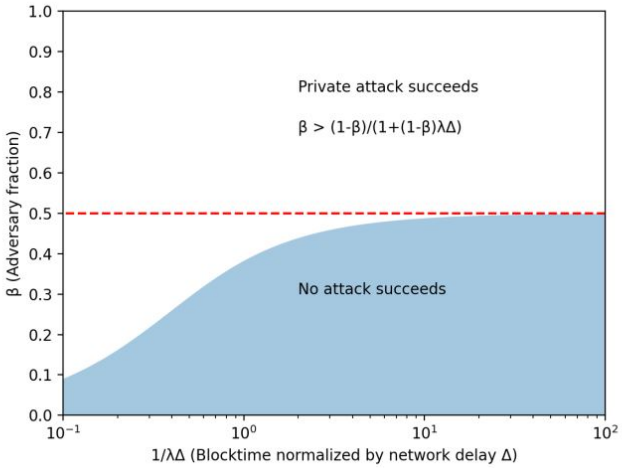


Figure 6.4: Minimum hash power needed by the adversary to succeed in its private attack. It turns out that this is also the security threshold for the adversary to successfully launch any safety attack.

Figure 5 shows an example where the private attack is not the worst-case attack. In this scenario, the adversary can use a different strategy and still succeed in disrupting the safety of the blockchain, even though the private attack fails. This implies that the private attack is not always the most effective attack in the presence of network delay.
However, despite this, it turns out that the private attack is still a worst-case attack in the sense of the minimum required honest hash power to ensure safety against all attacks. In other words, if the blockchain system can withstand the private attack with a certain level of honest hash power, it can also withstand any other attack with the same level of honest hash power. This safety guarantee holds with probability approaching one as the confirmation depth (k) approaches infinity.
The reason why the private attack is still a worst-case attack in terms of the required honest hash power is due to the inherent properties of the Nakamoto consensus protocol. The protocol is designed such that an attacker needs to have a majority of the total hash power to overpower the honest nodes and control the longest chain. Therefore, the same condition derived for the success of the private attack also holds for any other attack, and the private attack remains a worst-case scenario for determining the minimum required honest hash power to ensure safety in the blockchain system.

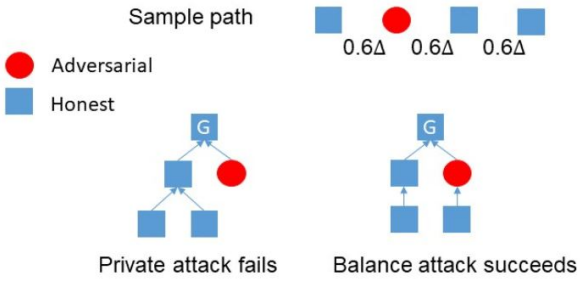


Figure 6.5: Attacks aimed to create a fork with length 2

6.9 Nakamoto blocks

The analysis of attacks on the blockchain system becomes more complicated when the network delay (δ) is non-zero, and the entire blocktree can be arbitrary due to multiple adversarial actions at different time instances. However, we can still gain insights by partitioning the complex blocktree into subtrees, each rooted at an honest

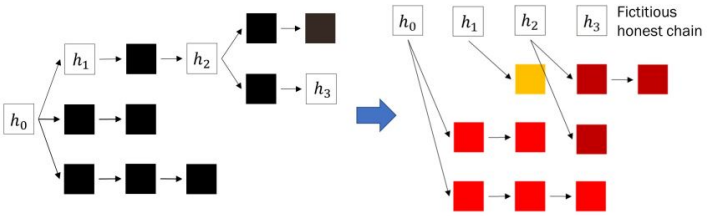


Figure 6.6: By blocktree partitioning, a general attack is represented as multiple adversarial chains simultaneously racing with a fictitious honest chain. Note that this fictitious chain is formed by only the honest blocks, and may not correspond to the longest chain in the actual system. However, the longest chain in the actual system must grow no slower than this fictitious chain.

block and consisting entirely of adversarial blocks (public or private).
By doing this, we can view the general attack as initiating multiple adversarial subtrees that race with a single fictitious chain consisting of only honest blocks (Figure 6). Each adversarial subtree's growth rate is upper-bounded by the growth rate of the adversarial chain used in the private attack. Therefore, if the private attack is unsuccessful, we know that the growth rate of each adversarial subtree must be less than that of the fictitious honest chain.
Under this condition, there must exist honest blocks, referred to as Nakamoto blocks, that have the special property that none of the past adversarial subtrees can ever catch up to the honest chain after the honest chain reaches that particular block (Figure 7). These Nakamoto blocks play a crucial role in stabilizing the blockchain. When a Nakamoto block occurs and occurs frequently, it guarantees the safety of the protocol.
The presence of Nakamoto blocks ensures that the prefix of the longest chain up to that block remains immutable in the future. In other words, once a Nakamoto block is included in the blockchain, it becomes a solid anchor that protects the entire prefix of the longest chain before that block from any adversarial reorganization attempts. This stability ensures that the blockchain system maintains its safety guarantees, and honest users can have confidence in the integrity of the ledger.
When Nakamoto blocks occur regularly, the blockchain's safety is effectively guaranteed, even in the presence of adversarial attacks. The analysis helps us understand how certain key blocks in the blockchain, such as Nakamoto blocks, play a critical role in maintaining the system's security and ensuring that honest users' transactions remain protected and valid.

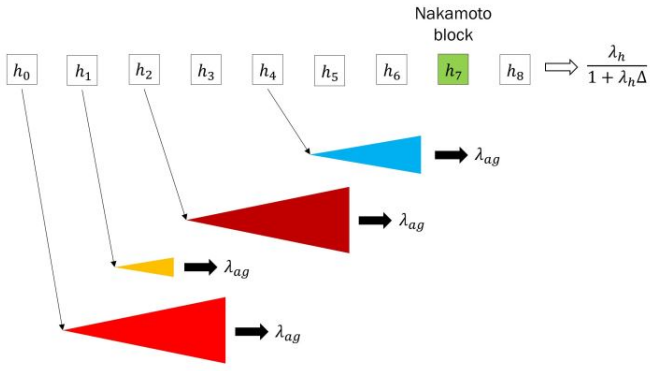


Figure 6.7: Race between the adversarial trees and the fctitious honest chain. While there may be multiple adversarial trees simultaneously racing with the honest chain, the growth rate of each tree is bounded by the growth rate of the adversarial chain in the private attack. An honest block is a Nakamoto block when all the previous adversarial trees never catch up with the honest chain past that block. To simplify notations, $\lambda_{ag} = \beta\lambda$ and $\lambda h = (1\beta)\lambda$.

6.10 Importance of synchronous network

The safety analysis of the longest chain protocol in this lecture highlights the importance of keeping the adversarial hash power within certain bounds relative to the overall mining rate and worst-case network delay. As long as the adversarial hash power is limited, the protocol remains safe, ensuring that the blockchain maintains its integrity and security.

However, the analysis also emphasizes the significance of having a finite worst-case network delay. In a purely asynchronous network, where there are no guarantees on message delivery times, the adversary can exploit the lack of synchronized communication to break consensus. By mining alternate chains and selectively sharing them with different subsets of honest nodes, the adversary can create confusion and disagreement among the nodes, leading to a breakdown of consensus and safety.

To address this issue, an intermediate network scenario, known as the partially synchronous network model, is considered. In this setting, the network delays are finite but unknown. There is a global stabilization time (GST), after which all messages are guaranteed to be delivered to all nodes within bounded time. In this partially synchronous setting, safety can potentially be violated because the confirmation depth k (used in the k-deep confirmation rule) could be smaller than the GST, allowing for vulnerabilities.

However, it is crucial to note that after the GST, consensus is restored, and the blockchain system becomes safe once again. This highlights the trade-off between the guarantees of safety and the network conditions in such scenarios.

In subsequent lectures, the focus will shift to studying consensus protocols that can ensure safety even under partially synchronous network settings, addressing the challenges posed by the unknown and varying network delays. These protocols aim to provide robustness and security even in more realistic network conditions, ensuring the integrity of the blockchain system under various scenarios.

Chapter 7

Liveness of Bitcoin

The last lecture focused on the safety of the longest chain protocol, which is a rule that determines which block is the valid one in case of a fork. The safety property means that once a block is confirmed by being k-deep in the chain, then the probability of it being deconfirmed by another fork is very small, as long as k is large enough. This ensures that the blockchain does not have conflicting histories, and that users can trust the transactions that are recorded in it.

We are interested in exploring the scenarios where the ledger does not receive any new blocks, or where some transactions from honest users are excluded from the blocks.

7.1 What is Liveness

Liveness is a term that describes the ability of a blockchain protocol to ensure that the transactions that are submitted by honest users are eventually included in the ledger, and that the blocks that contain these transactions are part of the longest chain. Liveness is an important security property because it guarantees that the network will not stop producing new blocks and confirming transactions, and that users can use the system without delays or failures. Liveness also prevents the network from being stuck in a fork, where different nodes have different views of the ledger.

Both liveness and safety are essential for a blockchain protocol to function properly and securely. A good consensus protocol should achieve both liveness and safety under reasonable assumptions about the network and the participants. However, there is often a trade-off between liveness and safety, as increasing one may decrease the other. For example, requiring more confirmations for a block to be considered final may increase safety, but it may also increase latency and reduce liveness. Therefore, finding the optimal balance between liveness and safety is a challenging task for blockchain protocol designers.

Liveness is definitely compromised if there is a deadlock. A deadlock occurs when the protocol gets stuck and the miners and parties cannot perform their tasks and mine new blocks.
Longest chain protocol cannot deadlock, which means that it will always produce a unique longest chain, even if there are multiple forks. This ensures that the network can always reach a consensus on the state of the ledger, and that users can always find the valid block.
The mining operation is very democratic, which means that any honest miner with a small amount of hash power can eventually succeed in mining a block and adding it to the chain. This encourages participation and decentralization in the network, and prevents monopoly or censorship by powerful miners.

There are two adversarial events that can threaten liveness:

1. All blocks on the longest chain are mined by an adversary
2. An adversary mines a block that is empty or censors specific transactions.

We need to examine Chain Quality, which is based on Chain Growth, to determine the likelihood of an honest miner's block being left out of the longest chain or under what circumstances this occurs.

7.2 Chain Growth

Definition 7.2.1: Chain Growth

Chain growth is defined as the rate of growth of the longest chain, denoted by CG.

If there is no fork in the chain, and only the longest chain exists, then its growth rate depends on the difficulty level, and it adds one block every 10 minutes on average. But it slows down for two reasons :

1. Not all miners may be behaving safely.
2. It could be that they're working in parallel

The chain keeps growing and new blocks are continuously added to the longest chain because of positive chain growth. An honest miner has a chance to mine a block, even if it is small, because the mining operation is random.

Chain growth depends on the behavior of the adversary, who controls a fraction of the hash power in the network, denoted by β. If the adversary stays silent, which means that he does not mine or broadcast any blocks, then chain growth is equal to the rate of block production by the honest nodes :

$$CG = \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}$$

One can see that the adversary really cannot do anything better than this. Delivering an honest block earlier than Δ time or publishing any adversarial block will just increase the chain growth. Therefore, we have the bound:

$$CG \geq \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta} \tag{7.1}$$

If the adversary acts honest, which means that he mines and broadcasts blocks according to the protocol rules, then chain growth is equal to :

$$CG = \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta} + \beta\lambda$$

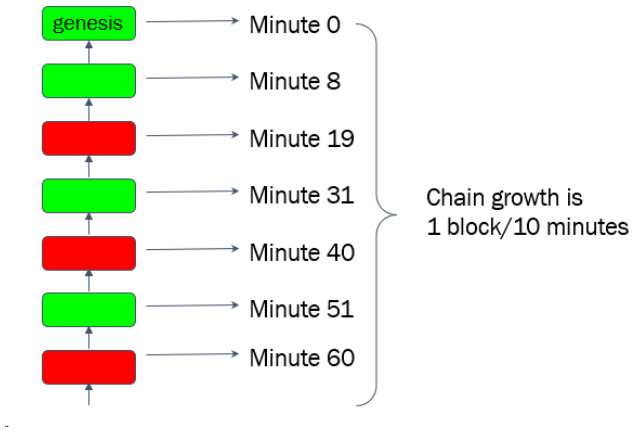


Figure 7.1: Chain Growth

7.3 Chain Quality

Definition 7.3.1: Chain Quality

We define the chain quality of a longest chain C as the fraction of honest blocks in C, denoted as CQ

$$CQ = \frac{\#\text{ of honest blocks in the longest chain}}{\#\text{ of all blocks in the longest chain}}$$

Positive chain quality ensures that a positive fraction of blocks mined by honest nodes enter the longest chain. Having enough blocks mined by honest nodes on the longest chain (i.e., high CQ) ensures that transactions enter the blockchain at a fast clip – thus CQ helps quantify the level of liveness.

Furthermore, CQ quantifies fairness: since block rewards are provided to blocks in the longest chain, having blocks mined by honest nodes in the longest chain ensures that honest miners are rewarded.

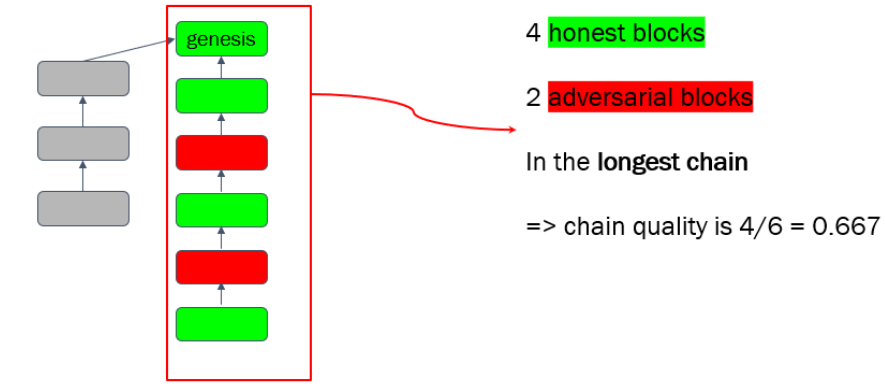


Figure 7.2: Chain Quality

Positive chain growth and positive chain quality together combine to ensure that any honest transaction will eventually be added to a block on the longest chain and to the ledger, which gives us liveness.

If the chain growth is CG and the adversary can mine blocks at a rate of at most $\beta\lambda$, by the definition of the chain quality, we have a lower bound:

$$CQ \geq \frac{CG \times T - \beta\lambda T}{CG \times T} = \frac{CG - \beta\lambda}{CG} \tag{7.2}$$

Plugging 7.1 in 7.2 :

$$CQ > 0 \Leftrightarrow \frac{(1-\beta)\lambda}{1 + (1-\beta)\lambda\Delta} > \beta\lambda$$

This is exactly the same threshold as the safety guarantee as we have seen before. Therefore, we can conclude the following security guarantee on the longest chain protocol.

Theorem 7.3.1

The longest chain protocol is safe and live, exactly under the following condition:

$$\frac{1-\beta}{1 + (1-\beta)\Delta} > \beta$$

7.4 Selfish Mining

The ideal chain quality is when the number of honest blocks on the longest chain is proportional to the honest hash power, which is $CQ = 1 - \beta$. However, this does not happen in the longest chain protocol because of **selfish mining**, which is a strategy where an adversary tries to create a private fork and withhold it from the network until he can overtake the public chain.

Consider the following adversarial strategy :

- The adversary always mines on the block at the tip of the longest chain, whether the chain is private or public. Upon successful mining, the adversary maintains the block in private to release it at an appropriate time.
- When an honest miner publishes a block the adversary will release a previously mined block at the same level (if it has one). We assume that the adversary can break ties in its favor, so honest miners will mine on the adversarial block.

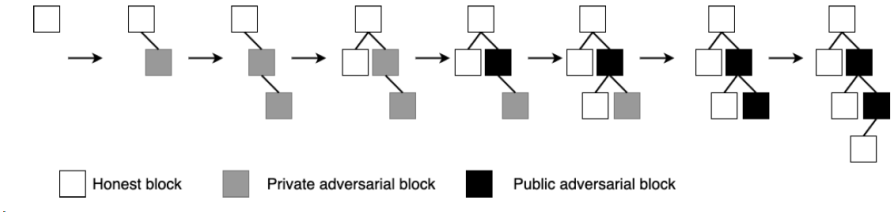


Figure 7.3: Selfish mining example

Selfish mining reduces the chain quality and liveness of the protocol, and gives an unfair advantage to the adversary.

7.5 Fruitchains

The longest chain protocol would have the best possible chain quality, $CQ = 1 - \beta$, if the adversary followed the protocol rules. But what if the adversary tries to cheat and disrupt the protocol? Can we design a better protocol that can achieve the optimal chain quality no matter what the adversary does? And can we do it with only a minor change to the longest chain protocol?
Fruitchains is a protocol that does exactly that. It achieves the optimal chain quality, and it does it fast.

The longest chain protocol links the security and the rewards of a block. The more blocks are built on top of a block, the safer and more profitable it is. Fruitchains is a new protocol that separates these two aspects by using two different types of blocks:

- **transaction block**, which has all the transactions (or data) that a block in the longest chain protocol would have.
- **proposer block**, which has only the header information that a block in the longest chain protocol would have.

These two types of blocks need to be linked and coupled together in a secure way. This is done by putting the hash of each transaction block inside a proposer block (see Figure **??**). The two types of blocks are also connected during the mining process by using "2 for 1 mining", also known as **cryptographic sortition**.

7.5.1 Cryptographic Sortition

In the mining process, the two types of blocks – transaction and proposer blocks – are mined together. A miner first creates a **superblock**, which is similar to a normal block in the longest chain protocol. The superblock has

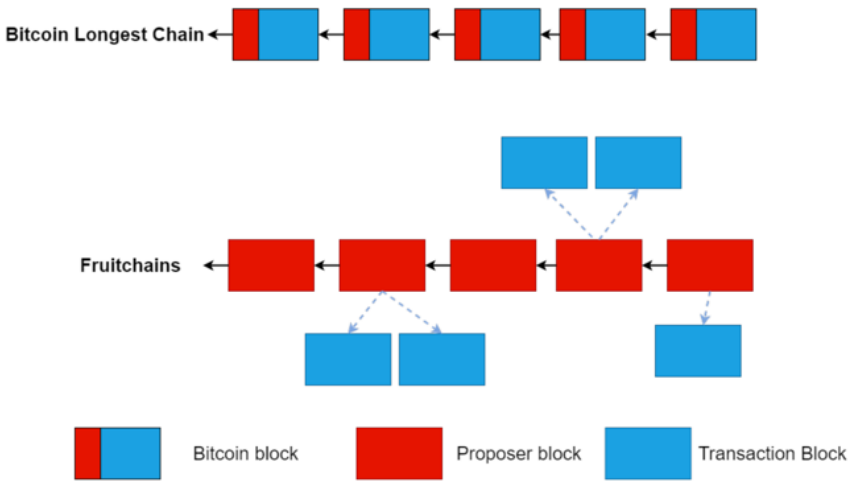


Figure 7.4: Comparison between blocks in the longest chain protocol and the transaction and proposer blocks in Fruitchains.

two parts: a transaction block and a proposer block. But that's where the similarity ends.
We say a superblock is successfully mined if

$$Hash(\text{nonce}, \text{superblock}) < T_{tx} + T_{prop}$$

The mining process determines whether a superblock is a transaction block or a proposer block based on the hash output:

- If the hash output is smaller than T_{prop}, the superblock is a proposer block.
- If the hash output is between T_{prop} and $T_{tx} + T_{prop}$, the superblock is a transaction block.

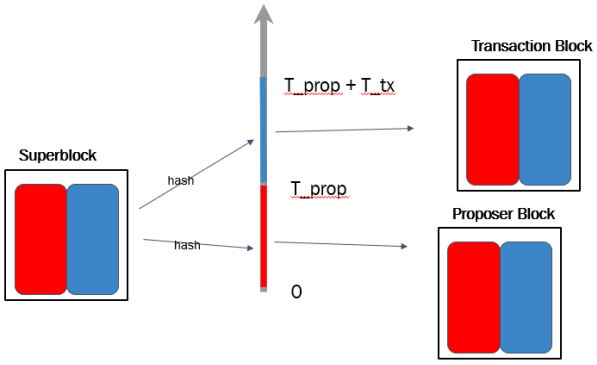


Figure 7.5: Superblock

The hash functions produce random outputs, so the chance of a superblock being a proposer or a transaction block depends on T_{prop} or T_{tx}, respectively. This means that two different kinds of blocks are mined together, but only one of them is chosen as the result of the mining process. This is called "two-for-one mining" or "cryptographic sortition".

7.5.2 Fruitchain Protocol

The ledger is a list of transactions that are sorted by some rules. The rules are:

- First, sort the proposer blocks by using the longest chain rule, which means that the valid chain is the one with the most blocks.
- Second, sort the transaction blocks that each proposer block refers to. For example, you can sort them by their hash value, which is a unique code for each block.
- Third, sort the transactions in each transaction block. For example, you can sort them by using the lexicographic ordering of the transactions in the Merkle tree representation, which is a way of organizing the transactions in a tree shape.

Safety of fruitchain protocol

Assuming λ_{tx} and λ_{prop} as the mining rate of the transaction blocks and proposer blocks respectively; these mining rates are directly proportional to the difficulty targets T_{tx} and T_{prop}, respectively. The safety of this protocol follows directly from the safety of the longest chain protocol: as long as λ_{prop} is small enough, i.e.,

$$\frac{1-\beta}{1+(1-\beta)\lambda_{prop}\Delta} > \beta$$

the k-deep proposer block in the longest chain will be permanent with high probability (for large enough k). This stabilizes the ledger guaranteeing safety.

Liveness of fruitchain protocol

We want to show that the optimal CQ of 1β is achieved no matter what the adversary does to interfere with the transaction and proposer block mining. We start by looking at the selfish mining attack: suppose an adversary tries to hide some proposer blocks that were mined by an honest party (and that contain some honest transaction blocks) by mining on a secret branch. Then, by the liveness property of the longest chain protocol (which guarantees positive CQ and CG), sooner or later an honest party will mine a new proposer block that includes those displaced transaction blocks. And if k is large enough, this proposer block will be securely added to the ledger.

We know that the honest and adversarial parties mine transaction blocks according to Poisson processes with rates $(1\beta)\lambda_{tx}$ and $\beta\lambda_{tx}$ respectively. And by the liveness property of the longest chain protocol, every honest transaction block will eventually be part of the proposer chain.Therefore, by considering an average on the entire longest proposer chain, we have

$$CQ \geq \frac{(1-\beta)\lambda_{tx}}{(1-\beta)\lambda_{tx}+\beta\lambda_{tx}} = 1-\beta$$

Note that here we define chain quality as the fraction of honest transaction blocks instead of honest proposer blocks, and further determining mining rewards by transaction block yields fair rewards.

7.5.3 Short time scale optimal CQ

The protocol is not secure in a short time span, because a private block attacker could secretly mine many transaction blocks and release them all at once, making a large portion of the proposer chain consist of adversarial transaction blocks.

This issue is resolved by making sure that a transaction block is attached to a proposer block that is close enough to the proposer block that includes it. Each transaction block has two parent blocks:

- A confirmed parent, which is a proposer block that has been stabilized/confirmed (i.e., k-deep block) and that the transaction block hangs from.
- A proposer parent, which should be the tip of the longest proposer chain.

Note that a proposer block also has a confirmed parent, because the two-for-one mining process links the transaction block mining and the proposer block mining.

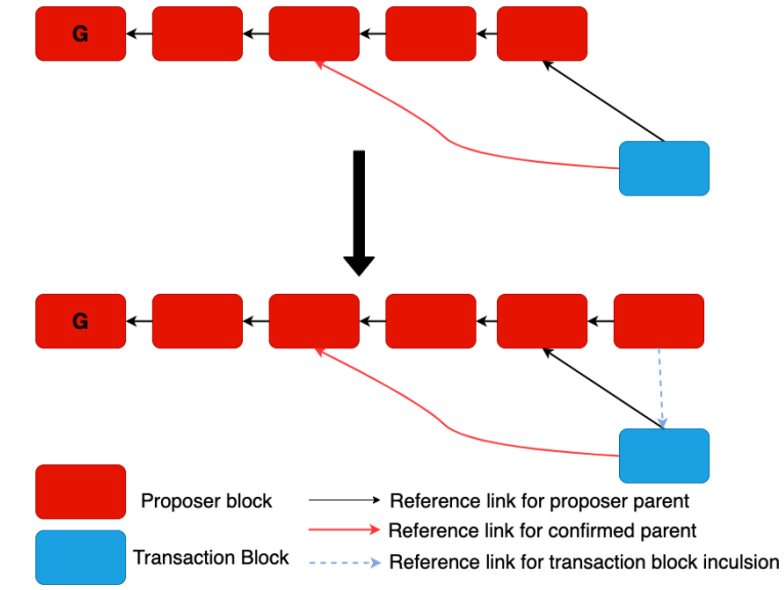


Figure 7.6: Fruitchain protocol

A transaction block B_{tx} is recent with respect to a proposer chain C if its confirmed parent is not too far from the tip of C. The distance is measured by the recency parameter R , which is a fixed number. We achieve our goal by only allowing proposer blocks to include recent transaction blocks. By choosing a large enough R, we make sure that any transaction block mined by an honest party will be deep enough in the proposer chain. So, the optimal CQ property still holds. Moreover, since only fresh transaction blocks can be included, the optimal CQ is achieved even for short parts of the longest chain.

Chapter 8

Scaling Throughput in Bitcoin

The main goal we are seeking to achieve in this chapter and the next following ones is to improve the performance of Bitcoin in different directions.
One measurement for the performance of blockchain platforms is the number of transactions per second (tx/s) that they can process. Bitcoin currently has a low tx/s rate, which limits its scalability and usability. Therefore, it is important to find ways to improve Bitcoin's performance in different dimensions.
Some of the dimensions to improve Bitcoin are:

- **Throughput**: how many transactions can be processed in a given time period.
- **Latency**: how fast a transaction can be confirmed and added to the blockchain.
- **Lightweight**: how much storage, communication, and computation resources are required to run the system.

There are two main approaches to improve Bitcoin's performance: layer one scaling and layer two scaling.

Layer one scaling refers to modifying the protocol itself, such as increasing the block size, reducing the block interval, or changing the consensus mechanism. These changes can increase the throughput and reduce the latency of Bitcoin, but they may also introduce security and compatibility issues.

Layer two scaling refers to building solutions on top of the protocol, such as payment channels, sidechains, or sharding. These solutions can enable faster and cheaper transactions without affecting the security and decentralization of Bitcoin, but they may also introduce complexity and interoperability issues.

8.1 Bitcoin's throughput is security limited

One of the challenges of Bitcoin is its limited transaction throughput. The transaction throughput of Bitcoin depends on two factors:

- **Block size** : The maximum amount of data that can fit in a block
- **Block interval** : The average time between two consecutive blocks.

Bitcoin can process about 7 transactions per second (tx/s), which is calculated as follows:

- A block is 1 MB, which is equivalent to 1,000,000 bytes.
- A transaction is about 250 bytes on average.
- Therefore, a block can contain about 4,000 transactions (1,000,000 / 250).
- A block interval is 10 minutes, which is equivalent to 600 seconds.
- Therefore, the transaction throughput is about 7 tx/s (4,000 / 600)

We can try increasing the two parameters λ,B to directly increase throughput:

- **Increasing** λ directly impacts the security. The hash power of the adversary that can be tolerated decreases as $\lambda\Delta$ increases
- **Increasing** B proportionally increases the network delay Δ: it takes longer to transmit a bigger packet

We conclude that the throughput of Bitcoin is limited by security constraints. Ethereum did it to the extent you thought that you can get away with the security risks.

To improve Bitcoin's throughput while keeping its security, three different methods were tried; the first two had some success but not enough, while the third solved the problem completely.

8.2 GHOST - embrace forking

When the mining rate or the block size increases, so does the product of λ and Δ. This product is related to the forking rate, which is the percentage of blocks that are left out of the longest chain.
A different mining rule that accepts forking (instead of rejecting it like the longest chain protocol) could be better for security. This is the idea behind the GHOST (greedy heaviest observed subtree) fork choice rule:
The "quality" of a chain is not measured by its height, but by how heavy the subtree that is linked to it is. That is the main idea.

The weight of a subtree is the total number of blocks in that subtree, regardless of whether they are valid or not. The rule then selects the chain that has the heaviest subtree as the longest chain.

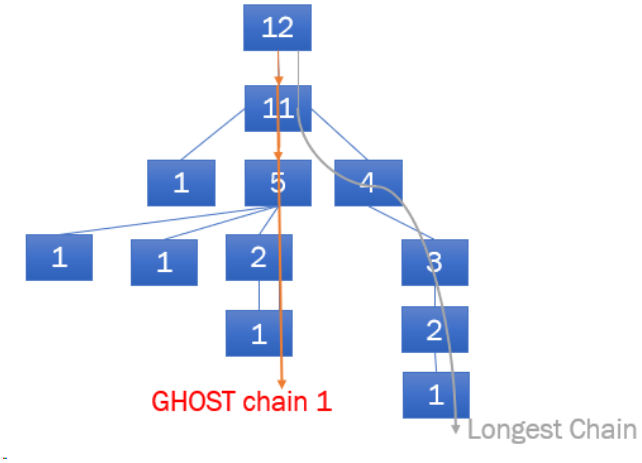


Figure 8.1: GHOST chain vs. Longest chain

In Figure 8.1, the weights of the blocks are labeled inside each block. The GHOST chain and the longest chain could be different.

8.2.1 Private attack on GHOST

One of the threats to the security of Bitcoin is a private attack. The GHOST rule claims to be secure against this attack, as long as more than half of the hash power is honest. The reason is that the weight of the honest subtree is proportional to the honest mining rate, while the weight of the private subtree is proportional to the adversary mining rate. Therefore, the honest subtree will always be heavier than the private subtree, and the GHOST rule will choose it as the valid chain. This security guarantee does not depend on the mining rate, so the throughput can be increased by increasing the mining rate, as long as the network can handle it.
Unlike the Longest chain protocol, private attack is no more the worst case attack for GHOST rule anymore.

8.2.2 Balance attack on GHOST

The balance attack aims to create two equally weighted chains (based on the GHOST rule) by the adversary. Honest miners choose the oldest chain to mine on when there are multiple chains with the same weight. The

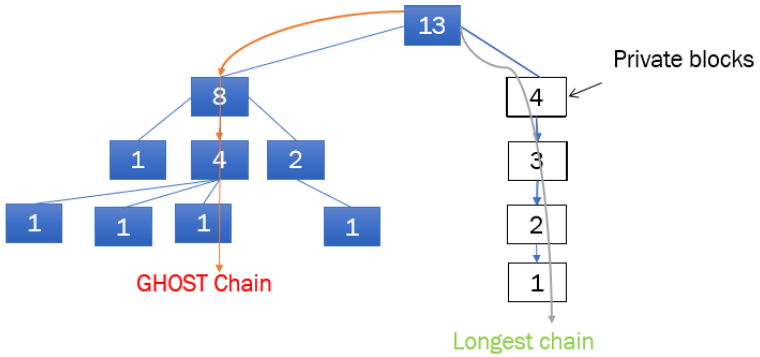


Figure 8.2: Private attack in GHOST

adversary can manipulate the network so that each half of the honest nodes see a different chain first. This way, the honest nodes are divided into two groups, each mining on a different chain, and the adversary can mine privately on both chains from the genesis block (see Figure 8.3). The adversary reveals its blocks as needed (to keep the balance of the two chains). The blockchain protocol becomes unsafe if the balance attack lasts longer than the confirmation depth k.

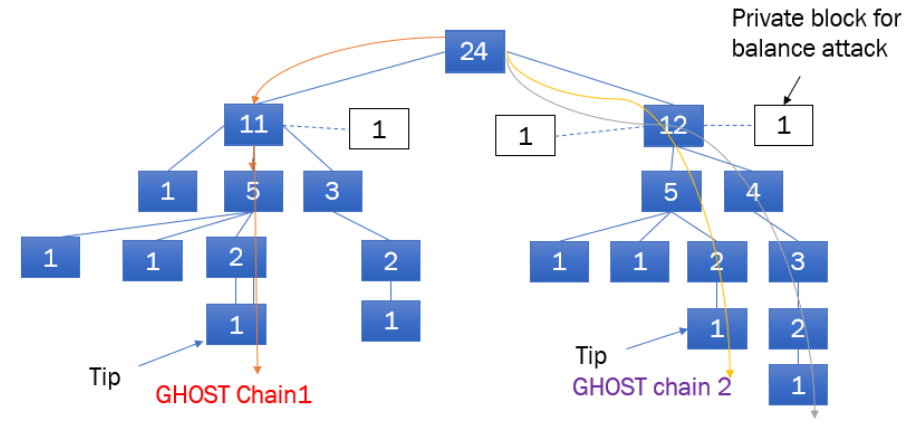


Figure 8.3: Balance attack on GHOST

8.3 Reduce Forking and Bitcoin-NG

One way to reduce forking in the blockchain is to modify the longest chain rule, which states that a miner can only propose one block for each successful nonce. Instead, we could allow a miner to propose multiple blocks for the same nonce, creating a longer chain faster. This would discourage other miners from forking the chain, as they would have to catch up with more blocks. However, this idea is different from simply increasing the block size.

Bitcoin-NG is based on the idea of dividing the blocks into two parts: a proposer block and a transaction block. The idea is somewhat like the Fruitchains protocol we learned in Chapter 7. This way, the proposer blocks are mined at a low frequency, but the transaction blocks are generated at a high frequency, allowing for faster confirmation and higher throughput of transactions.

The mining process is as follows:

- Unlike Bitcoin, where the miners remain completely anonymous and expose security risks, proposer blocks include the miner's public key when they are mined.
- The transaction blocks are not subject to proof of work and are simply signed (using the private key) by the

miner whose public key is embedded in the immediately preceding proposer block. Since the transaction blocks are not inhibited by proof of work, they can be added to the blockchain as quickly as the underlying network can support their reliable broadcast. The transaction blocks continue to be mined until a new proposer block appears on the longest chain, after which only this new proposer can mine transaction blocks.

- Proposer blocks and transaction blocks are interspersed with each other: proposer blocks are mined at the tip of the longest chain: the length is only defined via the number of proposer blocks in the chain.

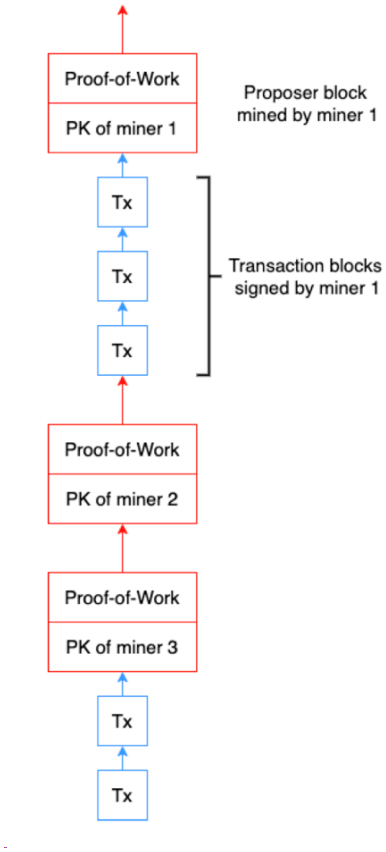


Figure 8.4: Bitcoin-NG mining process

8.3.1 Bribing attack in Bitcoin-NG

A bribing attack on Bitcoin-NG is a type of attack that tries to manipulate the incentives of the miners who create blocks on the blockchain. Bitcoin-NG is vulnerable to bribing attacks by an adaptive adversary, who can choose which miners to corrupt while the protocol is running.
A bribing attack is a way to incentivize the miners to join the adaptive adversary and collude with them.

The longest chain protocol has two properties that make it unpredictable and resistant to bribing attacks.

- First, no one can know in advance who will mine the next block, or what the contents of the block will be. This makes it hard for an adversary to manipulate the chain or the transactions.
- Second, once a block is mined, it is sealed by a nonce and cannot be changed without invalidating the proof-of-work. This makes it costly and risky for an adversary to bribe the miners to alter their blocks.

However, these two properties are only true for proof-of-work (PoW) blocks in Bitcoin-NG. They are not true for transaction (Tix) blocks. Tix blocks are not unpredictable, because they are identified by the public key of the

miner in the previous proposer block. Tix blocks are also not sealed, because they can be changed by the miner who has the private key associated with the previous proposer block.

One example of a bribing attack on Bitcoin-NG is the slow-down attack, which is not a security attack, but a performance attack. The slow-down attack aims to reduce the transaction throughput of Bitcoin-NG by stopping the generation of Tix blocks. The adversary can corrupt the miner who has the latest proposer block on the longest chain, and then bribe them to stop creating Tix blocks. This way, only the transactions in the proposer block will be confirmed, and the throughput will be reduced to the same level as Bitcoin. The slow-down attack can also make it easier for the adversary to perform double-spending attacks, because they can create more forks on the chain with fewer Tix blocks.

8.4 Prism 1.0

Prism 1.0 is a similar protocol to Bitcoin-NG, but it has some differences that make it more secure and efficient. Prism 1.0 also uses two types of blocks: proposer blocks and transaction blocks. However, in Prism 1.0, transaction blocks are not linked to each other or to the proposer blocks. Instead, transaction blocks are referred by proposer blocks, which means that proposer blocks contain the hashes of transaction blocks. This way, transaction blocks can be created and broadcasted independently, without waiting for the proposer blocks.

Another difference between Prism 1.0 and Bitcoin-NG is the proof-of-work (PoW) difficulty level for each type of block. In Prism 1.0, transaction blocks have a low PoW difficulty level, which means that they can be mined quickly and easily. This increases the transaction throughput and reduces the latency of the protocol. On the other hand, proposer blocks have a high PoW difficulty level, which means that they are mined infrequently and securely. This ensures the security and stability of the longest chain.

Prism 1.0 also uses a two-for-one mining procedure and cryptographic sortition to achieve unpredictability and resistance to bribing attacks. Due to the decoupling of transaction blocks from the security of the protocol, the throughput is only restricted by the capacity of the underlying P2P network. A full-stack UTXO implementation of Prism achieves more than 70,000 transactions/second.

The full Prism protocol also resolves another important limitation of the longest chain protocol: latency of confirmation.

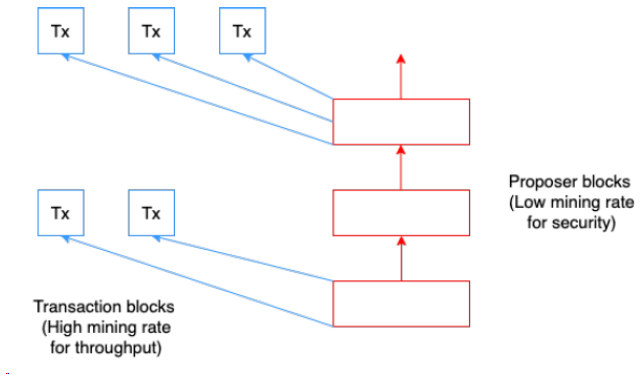


Figure 8.5: Prism 1.0 and Fruitchains protocols have two block types: proposer and transaction blocks, interspersed with each other. The transaction and proposer blocks are mined in parallel, coupled through the sortition process. Transaction block hashes are referred to by proposer blocks thus bringing the transactions into the confirmed ledger. Transaction block mining is kept easy (so throughput is high) and proposer block mining is kept difficult (so security is high).

Chapter 9

Scaling Latency

9.1 Bitcoin Latency

In Bitcoin, when a transaction has been included in a block, it will be confirmed only after the block is buried k-deep in the longest chain. Furthermore, to ensure confirmation with high probability, k must be large. The average latency of confirmation is then the product of k and average inter-block arrival time; again just like the throughput, the latency in Bitcoin is also limited by security. How bad is this trade off? in summary we have:

- **Confirmation in Bitcoin:** In Bitcoin, a transaction is confirmed when it is included in a block, which is then buried k-deep in the longest chain.
- **Latency of Confirmation:** The average latency of confirmation is the product of k and the average inter-block arrival time.
- **Trade-off with Security:** To ensure high probability of confirmation, k must be large. However, this trade-off is limited by security concerns.
- **Analysis with Zero Network Delay:** The analysis begins with the assumption of zero network delay ($\delta = 0$).
- **Error Probability (ϵ):** The largest probability of deconfirmation is achieved by the private attack. The error probability (ϵ) decays exponentially with the depth of the confirmation rule (k).
- **Exponential Decay of Error Probability:** The error probability (ϵ) is given by $\epsilon = e^{-ck}$, where $c = -log_e(4\beta(1-\beta))$.
- **Parameters:** Here, represents the fraction of the hash power of the adversary.

This analysis demonstrates that the error probability decreases exponentially with an increase in the confirmation depth (k) and is influenced by the fraction of hash power controlled by the adversary (β). It illustrates the security implications and limitations of setting the confirmation depth for transactions in the Bitcoin blockchain. In summary, the latency of Bitcoin's confirmation process is influenced by the error probability ( ) and the fraction of hash power controlled by the adversary ( ). The formula for the latency (τ_{LC}) is given as:

$$\tau_{LC} = \frac{k}{(1-\beta)\lambda} = \frac{1}{c\lambda(1-\beta)ln(\frac{1}{\epsilon})} = O(\frac{1}{\lambda ln(\frac{1}{\epsilon})})$$

where the constant hidden in the $O(\mathring{u})$ only depends on β. The latency increases as the error probability decreases, emphasizing the trade-off between security and confirmation time. For example, with $\beta = 0.3$ and $\epsilon = 10^{-3}$, the latency (τ_{LC}) works out to be approximately $\frac{66}{\lambda}$, which translates to around 11 hours when the average inter-block time ($\frac{1}{\lambda}$) in Bitcoin is 10 minutes.
The plot in Figure 1 illustrates this trade-off between latency and security, showing that even for modest security requirements, the latency can be significant in Bitcoin due to the potential for private attacks, which are the worst-case attacks for zero network delay ($\delta = 0$).

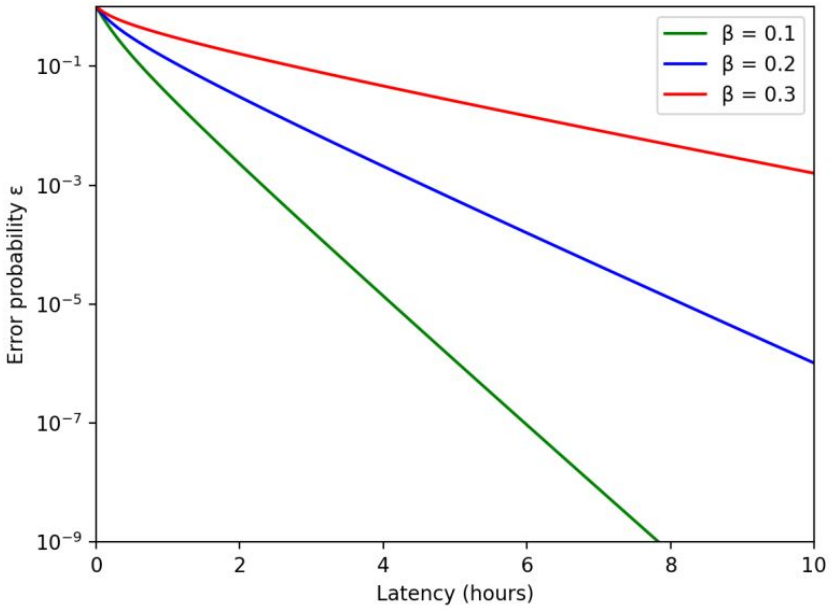


Figure 9.1: Bitcoin's latency–security trade-off for different β assuming $\delta = 0$. We set λ to be 6 blocks per hour, as in Bitcoin.

In the general case when $\delta > 0$, the private attack is no longer the worst-case attack in terms of success probability. However, it still provides a lower bound on the error probability, which consequently sets a lower bound on the latency for a given error probability. Both the error probability and latency tend to increase as δ increases due to increased forking caused by propagation delays in the network.

From a practical perspective, what is more interesting are the upper bounds on the error probability and latency, as they provide practical guidance for users of Bitcoin to determine when a transaction can be considered confirmed with a desired level of confidence. Recent research has derived such upper bounds, which are fairly close to the lower bounds implied by the private attack.

For example, consider a scenario where the adversary controls 10% of the total mining power and the block propagation delays are within $\delta = 10$ seconds. With these settings, a Bitcoin block can be confirmed with less than 10^{-3} error probability after approximately 5 hours and 20 minutes. For an even higher level of confidence, a block can be confirmed with less than 10^{-10} error probability after approximately 12 hours and 15 minutes.

Figure 2 illustrates the security-latency trade-off as implied by both the lower and upper bounds. The plot shows how the error probability decreases with increasing latency and how the upper and lower bounds are relatively close, providing a reasonable estimate of the confirmation time for a desired level of security.

9.2 Prism: Fast Confrmation

Lecture 8 introduces Prism 1.0, a blockchain protocol with two types of blocks: transaction blocks and proposer blocks. These blocks separate the security and payload aspects of the longest chain protocol. The proposer blocks serve two roles: they act as leaders for the "round" and provide confidence to ancestor blocks through the k-deep confirmation rule. Additionally, they vote for all ancestor blocks via parent link relationships. The full Prism protocol further decouples voting from proposer blocks by introducing separate voter blocks. The proposer blocktree anchors the blockchain, and each proposer block contains reference links to transaction blocks and a reference to its parent proposer block. Honest nodes mine proposer blocks following the longest chain rule in the proposer tree. The level of a proposer block represents its distance from the genesis proposer block, while the height of the proposer tree is the maximum level containing any proposer blocks. To determine the ordering of

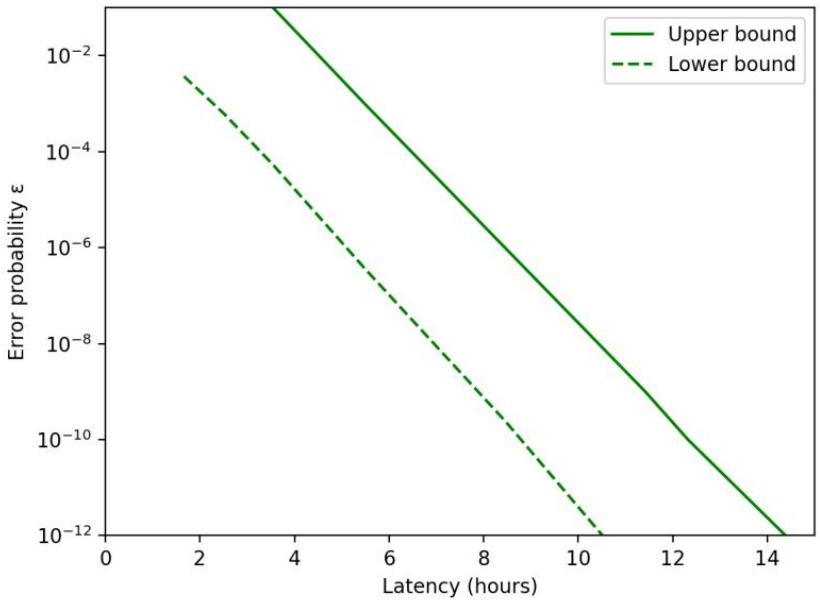


Figure 9.2: Bitcoin's latency–security trade-off for $\beta = 0.1$ when $\delta > 0$. We set λ to be 6 blocks per hour as in Bitcoin and $\delta = 10$ seconds. The lower bound on latency is derived from the private attack, while the upper bound is borrowed from this work.

proposer blocks and transactions, one leader proposer block is elected from each level, forming the leader sequence up to the height of the proposer tree, determined by the voter chains.

- Prism 1.0 introduces two types of blocks: transaction blocks and proposer blocks.
- These blocks decouple the security and payload aspects of the longest chain protocol.
- Proposer blocks have two roles: proposing as leaders for the round and adding confidence to ancestor blocks through the k-deep confirmation rule.
- Proposer blocks also vote for all ancestor blocks through parent link relationships.
- The full Prism protocol further separates voting from proposer blocks by introducing separate voter blocks.

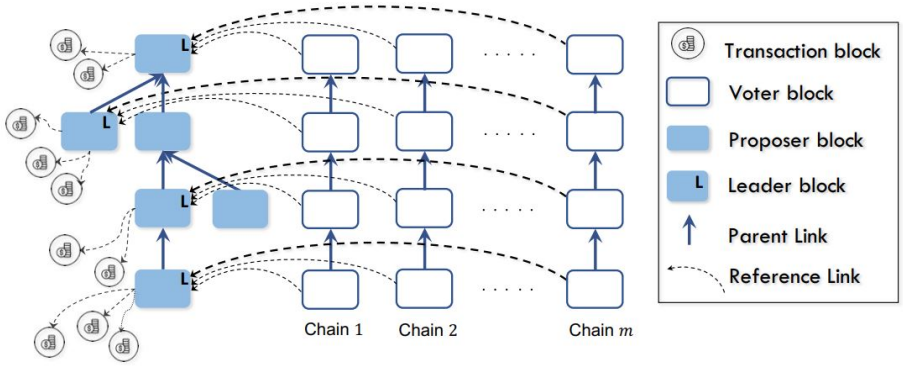


Figure 9.3: Factorizing the blocks into three types of blocks: proposer blocks, transaction blocks and voter blocks.

- The proposer blocktree anchors the blockchain and contains reference links to transaction blocks and a parent proposer block reference.
- Honest nodes follow the longest chain rule to mine proposer blocks in the proposer tree.
- The level of a proposer block represents its distance from the genesis proposer block, and the height of the proposer tree is the maximum level with proposer blocks.
- One leader proposer block is elected from each level to determine the ordering of proposer blocks and transactions.
- The leader sequence, determined by voter chains, comprises leader proposer blocks up to the height of the proposer tree, independent of the chain structure of proposer blocks to avoid deadlocks.

In Prism, there are m voter chains, where $m1$ is a fixed parameter set by the system designer. The value of m determines the level of parallelism in the voting process, and a larger m leads to shorter confirmation latencies. The choice of m is typically limited by network bandwidth and memory management considerations. For example, a full-stack implementation of Prism may use $m = 1000$.
New voter blocks are continuously mined on each voter chain, following the longest chain rule. Each voter block serves as a vote for a proposer block by containing a reference link to that proposer block. The voting process has specific requirements:

1. A vote is valid only if the voter block is part of the longest chain in its respective voter tree.
2. Each voter chain votes for exactly one proposer block at each level.
3. Each voter block votes for all proposer levels that have not been voted by its parent.

The leader block at each level is determined by the proposer block that receives the highest number of votes among all proposer blocks at that level. In case of ties, the leader is chosen based on the hash of the proposer blocks. The elected leader blocks establish a unique ordering of the transaction blocks, forming the final ledger of the Prism blockchain.
In Prism, cryptographic sortition is employed to prevent the adversary from concentrating its mining power on specific types of blocks or voter chains. To form a "superblock," a miner combines three components: a transaction block, a proposer block, and a voter block from the i-th voter tree, where $1im$. A superblock is considered successfully mined if its hash value, computed using a nonce, is less than the threshold value $T_{tx} + T_{prob} + mT_v$. Once a superblock is successfully mined, it is identified as one of the following based on its hash output. (See Figure 4 for an illustration):

- A proposer block if the hash output is less than T_{prop}.
- A transaction block if the hash output falls within the range $[T_{prob}, T_{tx} + T_{prob})$.
- A voter block on the i-th voter tree if the hash output falls within the range $[T_{tx}+T_{prob}+(i1)T_v, T_{tx}+T_{prob}+iT_v)$, where $1im$.

As we said above, Prism is a blockchain protocol that decouples the security and payload aspects of the longest chain protocol. It introduces two types of blocks: transaction blocks and proposer blocks, which act as leaders and voters in the security aspect. Additionally, there are m voter chains to parallelize the voting process, and cryptographic sortition prevents the adversary from concentrating mining power on specific blocks or voter chains. The protocol ensures a unique ordering of transaction blocks and enables ledger sanitization after the ledger has stabilized.

More details are listed below:

1. Prism employs cryptographic sortition to prevent the adversary from focusing its mining power on specific blocks or voter chains.
2. Superblocks are formed by combining a transaction block, a proposer block, and a voter block from the i-th voter tree ($1im$).

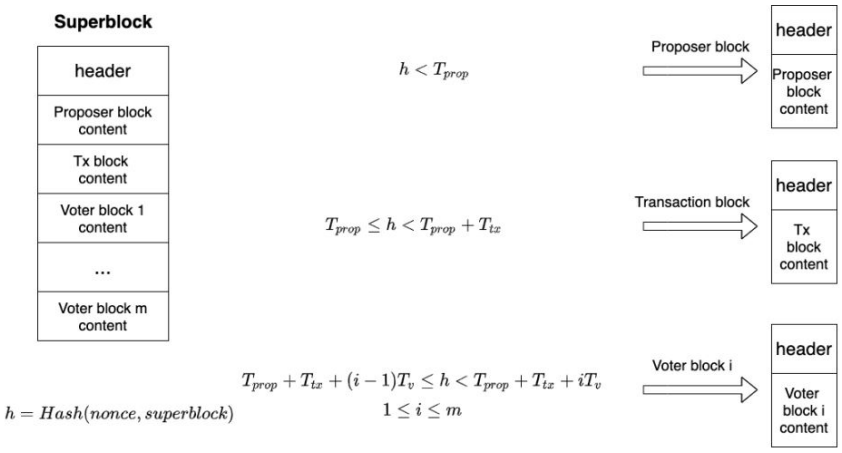


Figure 9.4: Superblock containing a transaction block, a proposer block and m voter blocks is "$(m+2)$-for-one" mined; the resulting cryptographic sortition identifes the type of the block as the winner of the mining process.

3. The successful mining of a superblock is determined by the hash output being less than specific threshold values T_{prob}, T_{tx}, and T_v.
4. The probability of a superblock being identified as a proposer block, a transaction block, or a voter block on the i-th voter tree is proportional to T_{prob}, T_{tx}, and T_v, respectively.
5. The transaction target difficulty (T_{tx}) is set easy, allowing for parallel mining of transaction blocks, while the proposer and voter chain target difficulties (T_{prop} and T_v) are set high for security.
6. The Prism protocol decouples the process of mining blocks from validating transactions, as the miner cannot validate a block before mining due to the non-chain structure of proposer blocks.
7. After a confrmed leader sequence of proposer blocks is established, the ordering of transaction blocks becomes uniquely determined, enabling ledger sanitization and validating individual transactions.

Overall, Prism combines parallelism, security, and decoupling to achieve efficient and secure transaction processing in a blockchain protocol.

9.3 Fast confrmation rule in Prism

In summary, Prism's parallel voting scheme allows for faster confirmation latency. By leveraging the law of large numbers, votes can be considered even if they are not fully stabilized, providing security while minimizing latency. However, these techniques need to be carefully analyzed to ensure the protocol's robustness against attacks. More details are listed below:

1. **Parallel Voting Scheme in Prism:** The parallel chain structure of the voting scheme in Prism allows for low confirmation latency. Votes accrue in parallel, unlike Bitcoin's sequential voting approach, leading to faster confirmation times for a fixed level of security.
2. **Simple Case of Confrmation Latency:** Let's consider a simple case in Prism where a single honest proposer block (B_p) at level one is mined at time 0. In Prism, B_p can be confirmed if no other proposer block at level one receives more votes than it. However, this naive solution is not operational since the confirmation depends on the future.
3. **Principled Confirmation Rule:** A principled confirmation rule that balances security and fast latency is needed. Waiting until each vote stabilizes by being k-deep in their chains (similar to Bitcoin) results in high latency. However, it's not necessary to wait for each vote to stabilize fully.

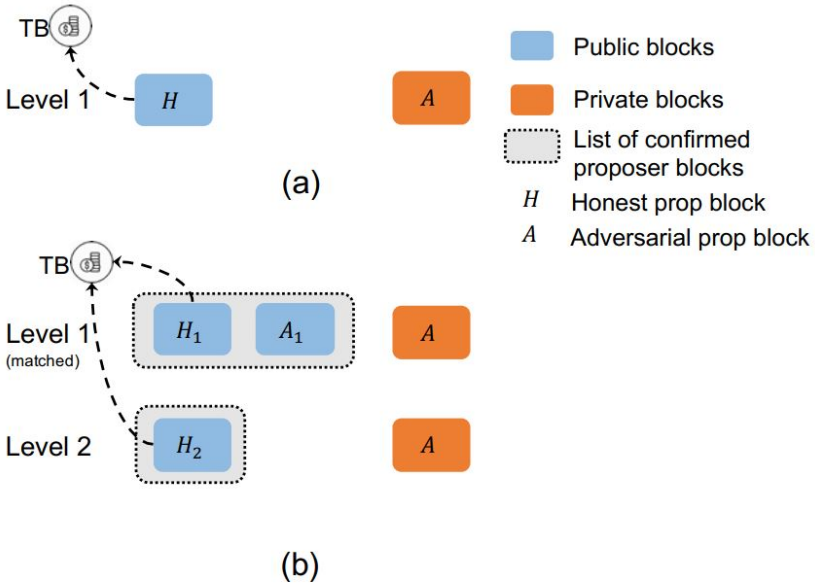


Figure 9.5: (a) Transaction block is referred to by an isolated honest proposer block. (b) Transaction block is referred to by a non-isolated proposer block but on the next level there is an isolated proposer block. Note that since H_2 is honest, it refers to all unconfrmed transaction blocks, i.e., TB.

4. **Leveraging the Law of Large Numbers:** Even if each vote is ephemeral (with a small value of k, e.g., $k = 2$ or 3), the averaging of all votes ensures overall security. This phenomenon is analogous to the law of large numbers or boosting in machine learning, where combining many low-precision classifiers creates a high-precision classifier.

5. **Analysis of a Simple Attack:** Consider a scenario where an honest proposer block H refers to a transaction block TB and is currently isolated at its level, meaning no other public proposer block exists at the same level. Block H starts collecting votes, each from the longest chain of its respective voter tree (See Figure 5(a)). The adversary attempts a private attack by mining a private proposer block A at the same level and forking off private alternate chains on each voter tree. It then sends its vote to block A. The adversary continues to mine on each voter alternate chain to overtake the public longest chain and shift the vote from H to A. The attack is successful if the adversary gets more votes on A than on H.

9.3.1 Fast confrmation rule

The use of multiple voter chains in Prism reduces confirmation latency while maintaining security. By leveraging Nakamoto's calculations and employing averaging principles, the protocol achieves faster confirmation times without compromising on robustness against attacks. In summary we can say:

1. **Attack Analogy to Nakamoto's Private Attack:** The attack in Prism, with m chains, can be seen as the multi-chain analog of Nakamoto's private attack on Bitcoin. Instead of a single race between the honest chain and the private chain, we have m races in Prism.

2. **Using Nakamoto's Calculations for Determining Waiting Depth:** Nakamoto's calculations on the success probability of an attack on a single chain can help determine how deep we should wait for votes to confirm the proposer block H in Prism. For tolerable adversary power ($\beta = 0.3$), the reversal probability in a single chain is 0.45 when a block is 2-deep.

3. **Latency Reduction with Multiple Voter Chains:** With $m = 1000$ voter chains and each vote being 2-deep, the expected number of chains that can be reversed by the adversary is 450. The probability that the adversary can reverse more than half the votes (500) is about 10^{-3}. Thus, to achieve $\epsilon = 10^{-3}$, we only

need to wait for 1000 votes each 2-deep, resulting in much shorter latency than waiting for 24 block depths for each vote to be reversed with a probability of 10^{-3}.

4. **Conceptual Similarity to Averaging Classifiers:** The reduction in latency in Prism is conceptually similar to averaging many unreliable classifiers to form a strong aggregate classifier. With more voter chains, individual votes need less certainty of permanence, leading to reduced confirmation time without sacrificing security.

5. **Ensuring Security:** Despite the reduced waiting depth for votes, each voter chain in Prism is still operating slowly enough to tolerate adversarial hash power (). This ensures security against attacks even with shorter confirmation times.

9.3.2 Tentative Confrmation rule

By employing the tentative confirmation rule based on the function h(t), Prism achieves a confirmation time that is observable and accurate even in the absence of direct time information in the blockchain. This allows for efficient and secure confirmation of proposer blocks in the protocol.

1. **Tentative Confirmation Rule:** To confirm proposer block B_p at time t, we look for a time t where the function $h(t) = \frac{1}{2}$. The function $h(t)$ calculates the fraction of stabilized votes (or effective vote from one voter chain) at time t.

2. **Closed Form Expression for $h(t)$:** The closed form expression for $h(t)$ can be derived in the appendix and depends only on the mining rate λ and the tolerable adversary hash power fraction β.

3. **Numerical Calculation:** For instance, when $\beta = 0.3$, we can numerically calculate $t\frac{3}{\lambda}$, resulting in a confirmation time that is a small factor of the average inter-block arrival time.

4. **Time Estimation:** Time is not directly observable in a proof-of-work blockchain due to potential inaccuracies or adversarial manipulation of timestamps in blocks. However, it can be estimated from the growth of the voter chains in Prism.

5. **Accurate Estimation:** The large number of voter chains in Prism ensures accurate estimation of time even for short periods due to the law of large numbers. The total mining rate of all voter chains is $m\lambda$, and on average $m\lambda$ voter blocks are mined within t units of time.

6. **Block-Based Confirmation Rule:** Using the conversion from time to blocks, a block-based confirmation rule becomes observable and practical.

9.3.3 Final confrmation rule

Exactly, in Prism, the tentative confirmation rule states that proposer block B_p will be confirmed when $mt^*\lambda$ voter blocks are mined and there is no competing proposer block at the same level. As mentioned earlier, t is a function of the mining rate λ and the tolerated adversary hash power fraction β. For example, when $\beta = 0.3$, it is numerically calculated that $t\frac{3}{\lambda}$.
Therefore, under these conditions, Prism confirms the proposer block B_p once $3m$ voter blocks are mined, ensuring a secure and efficient confirmation process. The large number of voter chains (m) and the ability to estimate time accurately through voter chain growth contribute to the reduction in confirmation latency while maintaining security against adversarial attacks.

9.4 Non-isolated Proposer Block

Consider now the case when the transaction block TB is referred to by an honest proposer block H_1 which is not isolated at its level, i.e. H_1 is matched by an adversarial public proposer block A_1 (the competing proposer block could also be honest). This matching could persist for L levels until reaching a level when there is an isolated honest proposer block. See Figure 5(b) for the special case of $L = 1$. Let us separately consider the life cycle of an honest transaction vs. a double-spent one.

9.4.1 Honest transaction

For confrmation of an honest transaction, a naive approach would be to wait until we can definitively confirm either H_1 or A_1. However, this approach may be slow due to adversarial attacks attempting to balance votes between H_1 and A_1. A key insight is that for honest transactions, we don't need to know which of H_1 and A_1 will be confirmed—only that one of them will be confirmed. This weaker form of list confirmation works because if A_1 eventually gets confirmed, a later honest proposer block can still refer to TB.
To confirm an honest transaction at level i, we need two events:

1. **List confirmation of all levels up to i:** Once we have list-confirmed a set of proposer blocks at level i referring to TB (e.g., either H_1 or A_1 will be the leader), we know that no other block can be the leader at that level. However, list confirmation alone is not enough if the transaction is not present in all ledgers. In that case, we also need to wait for an isolated honest proposer level where the proposer block will include TB in the ledger.

2. **Isolated honest proposer level:** Once this isolated honest proposer level is confirmed, and all the preceding levels are list-confirmed, we can be sure that TB will appear in the final ledger.

The confirmation latency is thus the maximum of two parts:

1. **List confirmation:** We quickly confirm that the adversary cannot produce a private block A with more votes than the votes of public blocks H_1 and A_1. The logic is similar to the case of an isolated honest proposer block discussed above, viewing the situation as a race between honest nodes voting for the public blocks H_1 or A_1 and the adversary voting for A. Adversarial actions (e.g., presenting H_1 to half the honest nodes and A_1 to the other half) can cause the number of votes to be evenly split between H_1 and A_1, which can slow down list confirmation, albeit not significantly.

2. **Isolated honest proposer level:** For example, in Figure 5(b), if we wait until level 2, we see an isolated public proposer block H_2, which can be fast confirmed. At this point, we know that the final leader sequence at levels 1 and 2 is either H_1, H_2 or A_1, H_2, both of which contain our honest transaction since H_2 refers to all previous unconfirmed transaction blocks.

9.4.2 Double-spent transaction

For confirmation of double-spent transactions, we require stronger conditions than those mentioned earlier. Instead of list confirmation, we need unique block confirmation, which confirms which block at a proposer level will be the ultimate leader. This is achieved once list confirmation occurs, and one of the list-confirmed blocks can be reliably declared the winner. If one of the public proposer blocks H_1 or A_1 gathers many more votes than the other block, then we can fast confirm a unique leader, even for double-spent transactions.
However, certain adversarial attacks, like balancing the votes on H_1 and A_1, can cause the number of votes to be evenly split between H_1 and A_1. In such cases, we cannot fast confirm a leader block. In this scenario, we must wait until every vote on H1 and A1 stabilizes, at which point either H_1 or A_1 is confirmed, and only one of the double-spent transactions is accepted. A content-dependent tie-breaking rule can be used to break ties after votes are stabilized, ensuring that the final decision on the accepted transaction is deterministic.

9.5 Scaling latency via Prism

The latency of Prism, denoted as τ_{Prism}, under the "normal path" (i.e., when there is a single proposer block for a level) is given by:

$$\tau_{Prism} = O(\frac{1}{\lambda}),$$

where λ is the mining rate. This latency is independent of the error probability ϵ when the parameter m (number of voter chains) is chosen to be sufficiently large. The constant hidden in the $O(\mathring{u})$ notation depends only on the fraction of tolerable adversary hash power β.
Compared to Bitcoin, Prism achieves faster confirmation because it only needs to wait until the effective vote $h(t)$ is greater than 0.5. The use of many voter chains in Prism allows the application of the law of large

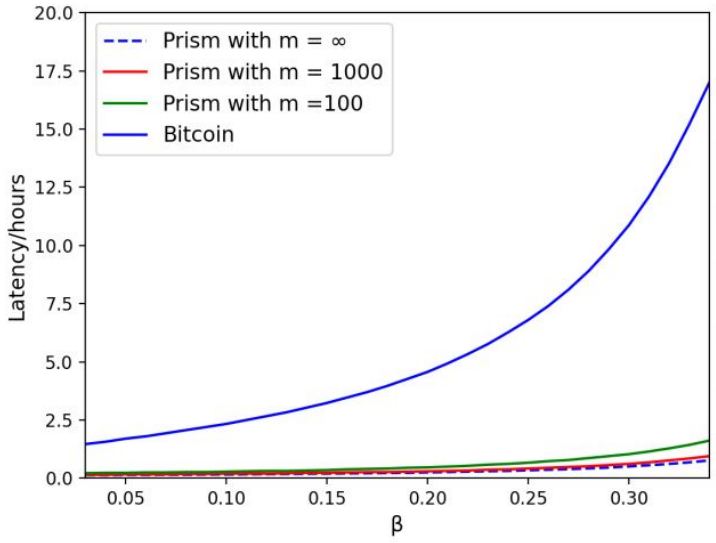


Figure 9.6: Comparison between Bitcoin and Prism latency under various values of β. We set λ to be 6 blocks per hour and the error probability $\epsilon = 10^3$. This also justifes the choice of m in the implementation of Prism.

numbers horizontally (in space). This means that the majority of the voter chains have voted for a proposer block irreversibly, and hence we can confirm the block with high probability.
In contrast, in Bitcoin, there is only one chain, so we have to wait until $h(t)$ is $1 - \epsilon$ to guarantee a small deconfirmation probability ϵ. In Bitcoin, the law of large numbers is applied vertically (in time), as discussed in Lecture 6.
The comparison between Bitcoin and Prism latency is illustrated in Figure 6. In summary, by using many voter chains, Prism eliminates the $ln(\frac{1}{\lambda})$ term present in Bitcoin's latency. This decouples latency from security and achieves fast confirmation.

Chapter 10

Sharding: Scaling Storage, Computation and Communication

10.1 Introduction

Lets tell two things: **Bitcoin uses full replication:** In traditional blockchains like Bitcoin, the number of participants does not affect the load per node significantly because everyone stores, validates, and communicates every block. This leads to linear growth in the total load of the network as the number of participants increases, which can be resource-intensive and costly.

However, in distributed systems, achieving horizontal scaling is desirable. Horizontal scaling means that the performance of the system, such as the number of transactions processed or files distributed, scales linearly with the number of participating nodes. An example of horizontal scaling is the BitTorrent protocol for file-sharing, where as more nodes join the network, the number of nodes storing a particular file increases, reducing the load per file request.

To achieve horizontal scaling in blockchains while maintaining consensus on the ledger order, the concept of sharding is used. Sharding involves splitting the database (ledger) into subsets, with each subset stored and managed by a different subset of nodes. This approach allows for full horizontal scaling, as not all nodes need to see and process all transactions. Instead, each node is responsible for a specific subset of transactions, allowing for better resource utilization and scalability.

The lecture focuses on the principled approach to sharding in blockchains, aiming to achieve full horizontal scaling without compromising on consensus and security. Sharding is a promising solution to address the resource burden and scalability challenges faced by traditional blockchain systems.

First order approach to Sharding In this first-order approach to sharding, the ledger is divided into K exclusive and non-intersecting subsets, each representing an independent application with its own transactions. These subsets are called shards, and the idea is to manage each shard as an independent blockchain. Each shard operates with its own consensus protocol, such as the longest chain protocol, and is maintained by N/K nodes out of the total N participating nodes in the network.

This approach achieves a form of horizontal scaling because each node is responsible for maintaining only one shard, which is a fraction 1/K of the entire ledger. Consequently, the maintenance costs, including storage, computation, validation, and communication, are reduced to 1/K of the overall ledger. As a result, the system can scale efficiently, and K can increase linearly with the number of participating nodes N, achieving optimal horizontal scaling.

However, this increase in efficiency and horizontal scaling comes at the cost of reduced security. The overall security of the system is limited by the security of any single shard, as an adversary can target and tamper with the state of the blockchain by attacking just one shard. Since each shard is maintained by a smaller subset of nodes (N/K nodes), attacking a single shard is easier for the adversary compared to attacking a single blockchain maintained by all N nodes.

In summary, the first-order approach to sharding provides K-fold horizontal scaling, but it decreases security by a factor of K. The challenge is to design a sharding mechanism that achieves horizontal scaling without compromising on security, which is the focus of the lecture. The lecture explores ways to provision horizontal scaling in blockchains while ensuring robust security for the entire system, even in the presence of adversarial

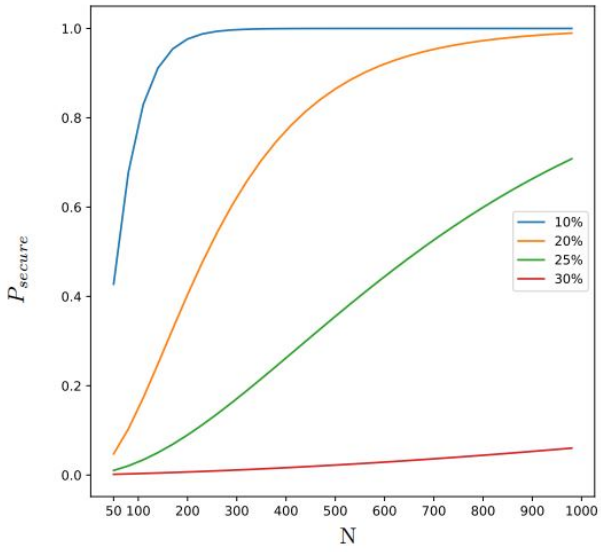


Figure 10.1: Probability that every shard is secure (i.e., honest supermajority in each of the $K = 10$ shards) as a function of the number of users N under different adversarial models (10%, 20%, 25% and 30% net adversarial fraction)

attacks.

10.2 Multiconsensus architecture

In this extended approach to sharding, the allocation of nodes to shards is done uniformly at random. An important aspect is that adversaries are no longer able to concentrate in a single shard to attack it. This is achieved through a cryptographically secure oracle that assigns nodes to shards randomly and fairly, and all participants can verify the correct allocation. This prevents adversarial collusion and maintains security even if a majority of honest nodes exists globally.
To enable random reallocation, a node-to-shard allocation engine (N2S) is introduced. The N2S is responsible for generating cryptographically secure random allocations of nodes to shards. This allocation can be managed by a separate service or as part of the blockchain itself. One approach is to use a distributed randomness generator like RandHound to create the randomness required for the allocation. Another method is to use a ”beacon” consensus engine shared across all shards, which generates a shared distributed randomness. The beacon only contains state commitments and N2S allocation metadata, making it lightweight and easy for all nodes to maintain.
While this solution achieves horizontal scaling, it comes with some limitations:

1. The N2S allocation requires each node to have an identity, making this sharding procedure not permissionless, which is different from Bitcoin’s design.

2. To maintain the fraction of honest hashing power in each shard and ensure global majority of honest hash power translates to majority honest hash power in each shard, the number of nodes per shard needs to be substantial, limiting the number of shards (K) as a function of the number of nodes (N), see Figure 1.

3. The architecture is not secure against an adaptive adversary, as the adversary can corrupt miners after they have been allocated to a shard, leading to collusion within a shard. To address this, regular reallocations of N2S allocations can be conducted to prevent coordination and collusion by the adversary. However, faster rotations of allocations may introduce more overhead in the N2S procedure, which diminishes the horizontal scaling effect.

These limitations are addressed in a full sharding solution, which is described next.

10.3 Uniconsensus architecture

In the uniconsensus architecture, the security properties of the multiconsensus approach are significantly improved by having every node participate in a single main consensus engine. This means that all nodes work together in the main consensus to maintain each shard, unlike the previous approach where only a small subset of nodes maintained a shard.
In this architecture, each node participates in a single main consensus engine, while also choosing a shard of its choice to mine shard blocks. This self-allocation of shards allows for better scalability and efficiency. The main consensus engine only maintains a log of the hash of shard blocks, which keeps it lightweight and easy for every node to maintain.
The coupling of shard blocks with the main consensus engine and ensuring adversary resistance is achieved through the same scaling principles we have seen in previous lectures. Specifically, the concept of "many-for-one mining" of a superblock and cryptographic sortition into constituent sub-blocks is utilized.
In other words, nodes work together to mine a superblock, which contains multiple shard blocks. This superblock is then cryptographically sorted into its constituent sub-blocks, each corresponding to a specific shard. This process ensures that shard blocks are securely integrated into the main consensus engine, and the cryptographic sortition adds a layer of protection against adversaries. In the uniconsensus architecture, the data structure consists of two main types of block structures: a proposer blocktree and K shard ledgers.

- **Proposer Blocktree:** The proposer blocktree is responsible for the single main consensus engine in the uniconsensus architecture. It is organized using a Proof-of-Work (PoW) based Nakamoto longest chain consensus protocol. Proposer blocks are the building blocks of this tree and contain the necessary information for the main consensus. Each proposer block also includes hash-pointers to shard blocks.
- **Shard Ledgers:** There are K shard ledgers, where K represents the number of shards. Each shard ledger consists of shard blocks. Shard blocks are specific to individual shards and are identified by their ShardID. These shard blocks contain shard transactions, similar to transaction blocks in the Prism protocol (as discussed in Lectures 8 and 9). Unlike the proposer blocktree, shard blocks do not contain any blocktree pointers, see Figure 2.

The absence of blocktree pointers in shard blocks is deliberate and serves a specific purpose. The ordering of shard blocks is solely inferred from their order in the main consensus engine, which is the longest chain of the proposer blocktree. This is similar to the way the Prism protocol determines the ordering of transaction blocks without relying on blocktree pointers. In the uniconsensus architecture, the main consensus engine's longest chain of proposer blocks is used to order shard blocks within each shard.
The primary responsibility of shard block mining in this architecture is to ensure adversary resistance, ensuring that the blocks remain secure against attacks. However, shard block mining does not play a role in determining the ordering of shard blocks. Instead, their ordering is entirely derived from their corresponding hash pointers in the proposer chain.

In this architecture, security is ensured by requiring every miner to mine both proposer blocks and shard blocks simultaneously. This is achieved through a process called "many-for-one mining" and cryptographic sortition.
Many-for-One Mining: Each mining node creates a superblock that contains a potential shard block for a specific shard of the node's choice, along with a potential proposer block. The miner then mines a nonce on the header of this superblock, which includes hash commitments of both the potential shard block and the potential proposer block. The miner's goal is to find a nonce such that the resulting hash of the superblock falls within certain predefined ranges.
Cryptographic Sortition: The cryptographic sortition algorithm ensures that the hash of the superblock falls within specific ranges, allowing it to be classified as either a proposer block or a shard block. Specifically, if the hash of the superblock falls between 0 and τ_1 (τ_1 represents the proposer block mining difficulty), then the superblock is treated as a proposer block. On the other hand, if the hash falls between τ_1 and τ_2 (where τ_2 is a shard-specific variable), the superblock is treated as a shard block for the identified shard.
The combination of many-for-one mining and cryptographic sortition prevents adversaries from selectively focusing their mining resources solely on mining either proposer blocks or shard blocks. This ensures a fair distribution of mining efforts and contributes to the security of the system.

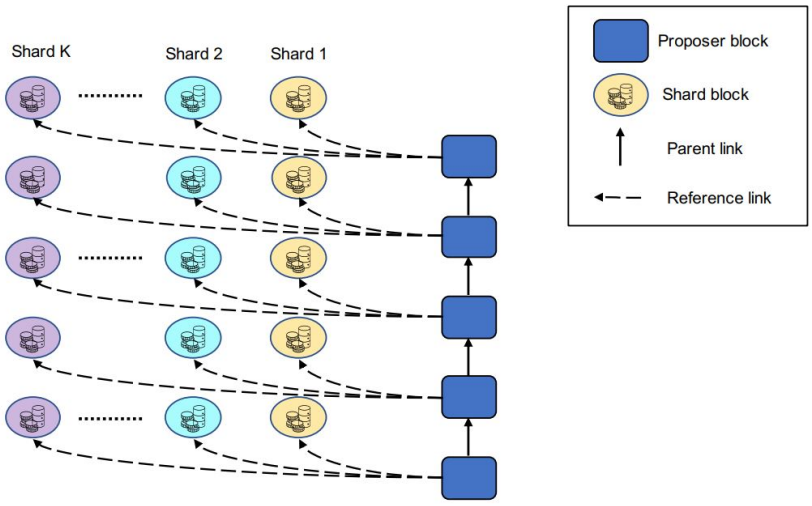


Figure 10.2: Data structures in the Uniconsensus architecture.

Moreover, during the mining process, the shard ID of the potential shard block is fixed, which means that a single mining attempt can only create a shard block for one specific shard (chosen by the miner). This helps maintain the integrity and separation of different shards in the system, This 2-for-one PoW sortition is shown in Figure 3. To maintain a consistent shard block generation rate for each shard, the shard block mining difficulty is periodically adjusted for each shard. This adjustment ensures that each shard generates blocks at a constant and controlled rate.

Indeed, the uniconsensus architecture offers significant improvements in resource usage and scalability compared to traditional full replication blockchains. By allowing each node to maintain only the proposer blocktree and a shard ledger of its choice, the system achieves better efficiency. Nodes are no longer burdened with maintaining the ledgers of all K shards, but only a fraction $\frac{1}{K}$, which reduces storage, computation, and communication requirements.
One notable advantage of the uniconsensus architecture is that it does not require a majority of honest nodes in each shard to maintain safety. This means that the system remains secure even when a shard has less than a majority of honest nodes. In such cases, adversarial nodes in the shard cannot alter the log of shard blocks without violating the safety properties of the consensus engine. Since all nodes participate in the main consensus engine, any malicious attempt to tamper with shard blocks would be quickly detected and rejected by the honest majority in the main consensus engine.
The elimination of the need for a consensus majority in each shard significantly strengthens the security of the uniconsensus architecture, making it safe against a fully adaptive adversary. Even with less than a majority of honest nodes in some shards, the overall security and integrity of the system are maintained, ensuring the resilience of the blockchain against various adversarial attacks.
The uniconsensus architecture addresses several limitations of the multiconsensus approach and offers several key advantages:

1. **Identity-free sharding:** Unlike the multiconsensus architecture, the uniconsensus architecture does not require an N2S allocation for nodes. Nodes are free to join any shard of their choice without the need for an on-chain validator/miner identity. This reduces complexity and allows for greater flexibility in node participation.

2. **Small number of nodes per shard:** Since the uniconsensus architecture does not rely on an honest majority within each shard, the constraint on the number of nodes per shard, as depicted in Figure 1, no longer applies. This allows for a more scalable and efficient allocation of nodes across shards.

3. **Adaptive adversary resistance:** The uniconsensus architecture is resilient to safety violations by corrupting a shard. Since the presence of an honest majority within a shard is no longer a requirement for

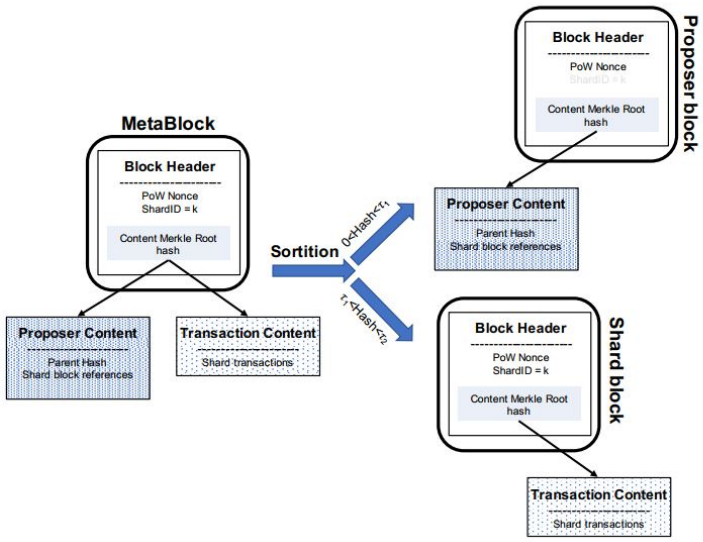


Figure 10.3: Uniconsensus architecture: 2-for-one PoW sortition.

security, destroying an honest majority in a shard is not a viable attack strategy. Additionally, the lack of an on-chain mapping of miner identity to shard ID weakens the targeting capabilities of adaptive adversaries.

However, the self-allocation of nodes in the uniconsensus architecture can lead to potential liveness attacks. An adversary could concentrate its mining power on a single shard, overwhelming honest shard blocks and reducing the fraction of honest blocks in the shard. This could throttle the throughput of honest transactions in the targeted shard, causing a loss in liveness.

To prevent such liveness attacks, a dynamic self-allocation (DSA) algorithm like Free2Shard can be employed. DSA allows honest nodes to respond to adversarial actions by relocating themselves to shards with low throughput, possibly due to liveness attacks. Such relocation is incentivized by higher transaction fees in shards with low throughput. By allowing nodes to dynamically self-allocate based on current conditions, DSA enhances the system's resilience to liveness attacks and maintains a more balanced and efficient distribution of nodes across shards.

10.4 Bootstrap and State-commitment

In both the multiconsensus and uniconsensus architectures, when an honest node decides to relocate to a new shard, it needs to efficiently download and verify the state of that shard. This process is akin to bootstrapping in a blockchain context. However, simply processing the entire ledger of the new shard would negate the scaling benefits of sharding since the node would end up processing the ledgers of all shards eventually.

To address this issue and enable efficient relocation to a new shard, the concept of trusted state-commitments is introduced. State-commitments are accumulators of a ledger's state that are universally agreed upon by all nodes in the network to be correct. They serve as cryptographic proofs of the validity of the ledger's state. When a bootstrapping node joins a new shard, it can download the latest state of that shard and verify its correctness using the state-commitments.

The accumulator used for state-commitments is a cryptographic data structure that efficiently allows for the inclusion of new data (transactions) while providing a proof that a particular element (state) is a part of the accumulator. By relying on these state-commitments, the bootstrapping node can trustlessly verify the state without relying on any single node in the new shard, ensuring security and integrity.

The process of generating and maintaining these state-commitments is crucial for the functioning of the sharding architecture. Properly designed state-commitments enable efficient and secure bootstrapping for nodes relocating to new shards, ensuring that the benefits of sharding in terms of scaling and efficiency are preserved.

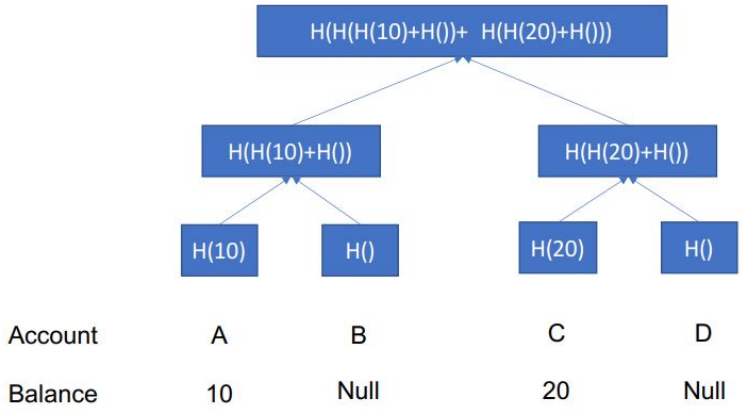


Figure 10.4: State-commitment using a sparse Merkle tree.

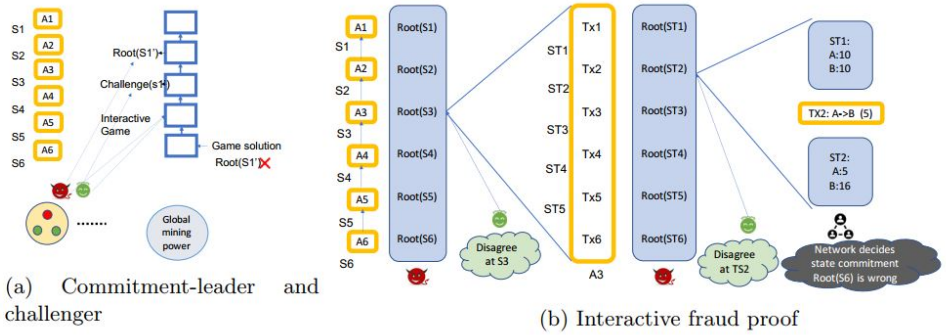
10.4.1 Accumulator for State-commitments

To design an accumulator for the (account, value) tuples in the ledger's state, a sparse Merkle tree is used. In a sparse Merkle tree, there exists a leaf in the tree for every possible output from a cryptographic hash function, representing every possible account number in the ledger. However, most of the leaves are empty, making the tree sparse. This allows for efficient simulations because only the non-empty leaves need to be stored and updated.
Unlike a regular Merkle tree, an ordered Merkle tree is not used for state-commitments because it is costly to add or remove accounts from the middle of the tree, especially when the tree is large.
In the sparse Merkle tree, the value of a leaf is the balance of the corresponding account if it exists or null if the account does not exist. This way, the tree can efficiently represent the dynamic nature of the ledger's state with insertions and deletions of accounts.
To further optimize the sparse Merkle tree, a compressed form called Merkle Patricia Trie (MPT) is often used. MPT has similar insertion and deletion complexity ($O(1)$) as a sparse Merkle tree but is more space-efficient, A sparse merkle tree with only two accounts is illustrated in Figure 4.
A state commitment consists of the root of the Merkle Patricia Trie representing a shard's execution state. This root is generated at regular intervals known as epochs and is posted on either the beacon chain (for multiconsensus architecture) or the main consensus engine (for uniconsensus architecture). By having all nodes agree on the state commitment's root, the validity and consistency of the ledger's state can be ensured across the network, enabling secure and efficient bootstrapping for nodes relocating to new shards.

10.4.2 Generation and agreement on state commitments

Ensuring that state commitments are generated and agreed upon in a Byzantine fault-tolerant manner is crucial for the security and integrity of the system. In both the multiconsensus and uniconsensus architectures, the process of posting state-commitments must be resistant to attacks and manipulations by adversaries. This focus on following discussion:

1. **Multiconsensus:** Indeed, the multiconsensus architecture benefits from the strong underlying assumptions of node identity and an honest majority within each shard. With these assumptions in place, the process of state-commitment generation becomes straightforward.
   Each shard in the multiconsensus architecture can generate a state-root representing the state of the shard at a particular point in time. This state-root is then signed by a majority of the nodes within that shard, effectively turning it into a state-commitment. The fact that a majority of nodes in the shard sign the state-root provides a strong level of trust and ensures the correctness of the state-commitment.
   State-commitments can be treated as regular transactions on the shard itself, subject to deterministic validation rules. This means that the execution state at the point in the ledger corresponding to the state-commitment must match the state represented by the commitment. This further reinforces the correctness



(a) Commitment-leader and challenger

(b) Interactive fraud proof

Figure 10.5: State-commitment in uniconsensus architecture.

and consistency of state-commitments.
To facilitate the availability of the latest state-commitments for incoming nodes from different shards, these commitments are periodically posted on the beacon chain. This ensures that all nodes in the network can easily access and verify the state-commitments without relying on centralized sources. Since state-commitments are small in size (usually just the output of a cryptographic hash), posting them on the beacon chain incurs minimal overhead.

2. **Uniconsensus** In the uniconsensus architecture, state-commitment generation is more intricate due to the absence of node identity and the lack of an assumption that each shard has an honest majority. Additionally, not all transactions included in the ledger are assumed to be valid, as validation is decoupled from ordering. This means that transaction validation occurs after the order of transactions has been agreed upon, making it challenging to verify the state-commitments across the entire network without defeating the purpose of sharding.
To address these challenges, the uniconsensus architecture incorporates two novel ideas: interactivity among the nodes generating state-commitments and relying on honest nodes to detect fraud in the broadcasts, rather than proving the veracity of each broadcast.
The state-commitments are generated by specific parties within a shard, known as commitment leaders. The broadcasts of these state-commitments are scrutinized for fraud by the honest nodes in the shard. If any fraud is detected, an honest node, referred to as the challenger, posts a proof of fraud on the main consensus engine, the proposer chain. If no fraud proof is posted within a specific time-frame, the state-commitment is assumed to be honest.
To keep the proposer chain lightweight, the fraud proof must also be lightweight. Since consecutive state-commitments summarize thousands of transactions, it is not desirable for the consensus engine to process all those transactions to verify a fraud-proof. Instead, the goal is to pinpoint the fraud proof of incorrect computation to just one transaction or bytecode. This is achieved through a logarithmic search across all transactions.
The process is illustrated in Figure 5a, where the fraud-proof is generated by a challenger and posted on the proposer chain to challenge the correctness of a state-commitment.
By employing interactivity and relying on honest nodes to detect fraud, the uniconsensus architecture establishes a fraud-proof based approach for state-commitments. Assuming correct if not proven wrong within a time-frame allows for an efficient and secure generation of state-commitments, even in the absence of node identity and an honest majority within each shard.

The logarithmic fraud search mechanism involves two parties, a commitment-leader, and a challenger, both belonging to the same shard. When the commitment-leader, who is adversarial, posts an invalid state-commitment on the consensus engine, the challenger detects the fraud and initiates the fraud search protocol. The following sequence of interactions summarizes the protocol once fraud is detected and challenged:

(a) The commitment-leader posts intermediate states for the challenged state, attempting to prove the validity of their state-commitment.

(b) The challenger responds with a number indicating the first intermediate state where their view differs from that of the leader.

(c) The protocol continues to the next round, with the commitment-leader posting intermediate states for the smaller challenged state, and the challenger responding based on the previous step.

(d) The interactions proceed in a recursive manner until the conflict is resolved down to a single transaction. At this point, the search for the single transaction is complete.

(e) The identified transaction is posted on the consensus engine to decide if fraud has occurred or not. If the transaction is indeed invalid, it is removed from the ledger, and consensus is regained.

The process is illustrated in Figure 5b, where the fraudulent state-commitment is challenged, and the protocol recursively searches for the specific transaction that caused the fraud.
Once fraud has been detected and the incriminating transaction is found, it is excised from the ledger, and consensus is restored. In some designs, the commitment-leader may also be penalized for this fraudulent behavior. For example, in a Proof of Work (PoW) setting, the penalty can be imposed outside the blockchain. In the context of Proof of Stake (PoS) mechanisms, the penalization can be handled within the blockchain, such as slashing a collateral transaction required to be posted by the commitment-leaders.
The fraud-proof mechanism, along with the ability to penalize adversarial behavior, incentivizes commitment-leaders to follow the protocol honestly, as they risk losing rewards or collateral if they engage in fraudulent activities. This approach ensures the integrity and security of state-commitments in the uniconsensus architecture.

10.5 Cross-shard transactions

The single-stage cross-shard transaction protocol presented above, while seemingly suitable for ensuring atomicity, does not actually achieve it due to a violation that arises in certain scenarios. Specifically, if funds x_B are not available in shard B, the protocol fails to ensure atomicity.
In this scenario, when the transaction is initiated, shard A successfully locks x_A and reflects this in its state commitment $S1_A$. However, shard B cannot lock x_B since those funds do not exist. As a result, a proof of lock from both input shards is not available for the cross-shard transaction. Consequently, output shards C and D do not release funds y_C and y_D. The fund x_A remains locked indefinitely, causing the transaction to be committed in shard A while being aborted in the rest of the shards. This inconsistency breaks atomicity.
To address this limitation and ensure atomicity in cross-shard transactions, a two-stage commit protocol can be utilized. The protocol involves the following steps:

1. Input shards (A and B in the example) initiate the transaction and lock the funds x_A and x_B for this cross-shard transaction. However, instead of directly updating their state commitments, they first create a provisional transaction record that indicates the intent to participate in the cross-shard transaction.

2. Output shards (C and D) receive the provisional transaction records from input shards and verify their validity, ensuring that the funds x_A and x_B are indeed locked.

3. Once the output shards are assured of the locks, they create their own provisional transaction records, indicating the receipt of funds y_C and y_D.

4. The output shards send a confirmation back to the input shards, acknowledging that they have successfully received the provisional transaction records and are ready to proceed.

5. Upon receiving the confirmation, the input shards finalize the transaction by updating their state commitments to reflect the lock of funds x_A and x_B.

6. Finally, the output shards also finalize the transaction by updating their state commitments to include the funds y_C and y_D.

By using a two-stage commit protocol, atomicity can be ensured in cross-shard transactions. The protocol ensures that if the transaction is committed (aborted) in one shard, it will be committed (aborted) in all participating shards, avoiding any inconsistencies and preserving the integrity of the overall transaction.

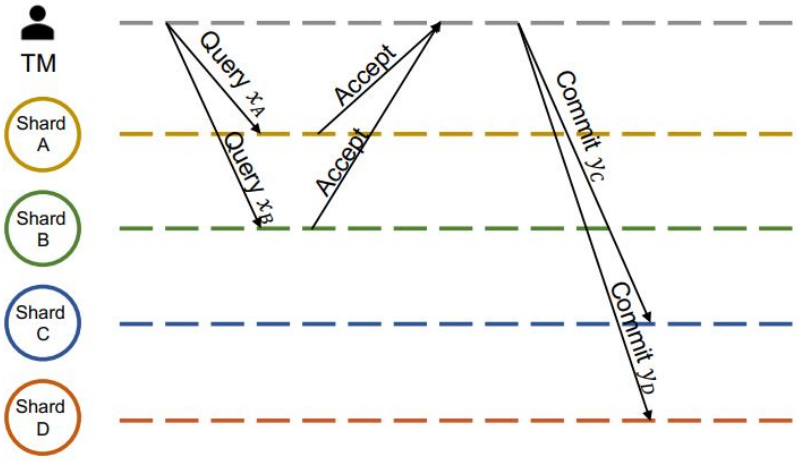


Figure 6: Cross-shard two-phase commitment.

Figure 10.6: Cross-shard two-phase commitment.

The generalized two-phase commitment protocol presented above ensures atomicity in cross-shard transactions by allowing a rollback of the transaction if any of the input shards fail to lock the required funds. The protocol works as follows:

Phase 1:

- A transaction manager (TM) is designated to read state commitments across shards. Typically, the TM represents the spending party/parties.
- If the input funds x_A and x_B are available in shard A and B, respectively, both shards update their state to lock the funds, indicating that they are reserved for the cross-shard transaction.
- If an input fund in shard A is not available (i.e., funds x_A are not present), shard A updates its state to mark fund x_A as unavailable by adding a receipt of unavailability to its state.

Phase 2:

- Once the TM confirms that the funds are locked in all input shards, as indicated by state commitments $S1_A$ and $S1_B$, it sends a proof of lock to the output shards (shards C and D).
- Upon receiving the proof of lock, the output shards generate the funds y_C and y_D for the cross-shard transaction and update their state commitments accordingly.

If the TM observes that funds are unavailable in any of the input shards (based on state commitments $S1_A$ and $S1_B$), it takes the necessary action to ensure atomicity:

- The TM sends a rollback transaction to all input shards, consisting of a proof of receipt of unavailability from one of the input shards.
- The rollback transaction prompts the input shards to release any locked funds (e.g., x_A) that were reserved for the cross-shard transaction.

By using this two-phase commitment protocol with a possibility of rollback, atomicity is guaranteed. The protocol ensures that either all the input funds of the transaction are successfully spent (locked) in their respective shards,

or the entire transaction is aborted, and any locked funds are reclaimed.
In the example scenario discussed earlier, where funds x_B are not available in shard B, the two-phase commitment protocol ensures that the transaction is aborted in all participating shards. Once shard A locks x_A and shard B marks x_B as unavailable, the TM can initiate a rollback transaction to unlock x_A in shard A, ensuring that the entire transaction is aborted consistently across all shards. This avoids any partial spending of input funds and maintains the integrity of the cross-shard transaction.

Chapter 11

Proof of Stake

11.1 Proof of Work is energy inefficient

The blockchains we have seen this far, from Bitcoin to its improvements, are based on the proof of work (PoW) mining procedure.
The design depends on the PoW process, which has several functions: it prevents spam on the network, it elects block proposers, and it protects against Sybil attacks on the protocol.

However, PoW mining is very wasteful of energy and hardware resources (GPUs). It has been estimated that Bitcoin's electricity consumption is similar to that of a mid-sized or large country. To put it another way, a single Bitcoin transaction uses as much electricity as 750,000 transactions with a credit card (such as Visa).
Moreover, as more people join the Bitcoin network, the mining competition increases and so does the energy usage. This is a negative network effect, where more participation leads to lower efficiency. Therefore, one of the most important aspects of scaling Bitcoin is reducing its energy consumption.

Therefore, one of the most important aspects of scaling Bitcoin is reducing its energy consumption. We will explore an alternative to PoW, called Proof of Stake (PoS), which aims to achieve this goal.

11.2 Proof of Stake

PoS and PoW are two different mechanisms for selecting block proposers in a blockchain protocol.
In PoW, block proposers are chosen based on their hash power, which is the amount of "work" they can do to solve the mathematical puzzle.

Proof of Stake (PoS) is a mechanism that allows participants to contribute to the blockchain protocol by owning coins, rather than by doing work. In PoS, block proposers are chosen based on their coin balance, which is the amount of stake they have in the network. The stake of each node is determined by the confirmed ledger, which is secured by the protocol.

PoS has several advantages over PoW, such as being more energy efficient and capital efficient. PoS does not require a lot of electricity or specialized hardware to run the protocol.

In the longest chain protocol, a miner succeeds in finding a nonce such that the following inequality is satisfied:

$$H(\text{prev hash}, \text{merkle root}, \text{nonce}) < T \tag{11.1}$$

where $H(\text{ů})$ is a cryptographic hash function, "prev hash" is the hash of the header of the previous (parent) block, "merkle root" is the Merkle root of the transactions in the block, "nonce" is an integer that can be adjusted by the miner, and T is a pre-defined threshold deciding the mining difficulty.

A PoS system starts with a set N of nodes that have some initial coins and keys. Each node $n \in N$ has a

coin with a stake value $stake_n$, and a pair of keys (pk_n, sk_n) for signing messages.
The first block of the system has all the public keys and stakes of all the nodes. We assume that the stake of a node stays the same and does not depend on the transactions in the blocks. This is called the static stake setting.

We now discuss how to design a leader election mechanism in the PoS system.

11.2.1 Idea 1

Definition 11.2.1: POS Idea 1

A node n succeeds in mining a block if

$$H(\text{hash(parent.header)}, \text{Merkle root of tx}, pk_n) < T \times stake_n \tag{11.2}$$

The equation is similar to PoW, except that it multiplies the threshold by the stake of the node. The idea is to make the probability of winning the stake lottery proportional to the stake of the node.
In order to eliminate the role of hash power, we should remove the "nonce" from the equation. The public key of the coin is also required in the hash function to prove the ownership of the coin.

However there is a problem with this idea, which is called grinding. Grinding means that a node can try different sets of transactions to change the Merkle root of the block, and thus affect the outcome of the equation. This way, the node can increase its chances of being selected as a block proposer, even if it has a low stake. Nodes equipped with better machines will have the advantage in winning this lottery.

Idea 2

This idea is to remove the transaction hash from the equation, so that the node cannot change the Merkle root to influence the outcome. This will protect the system against the grinding attack.

Definition 11.2.2: POS Idea 2

A node n succeeds in mining a block if

$$H(\text{hash(parent.header)}, pk_n) < T \times stake_n \tag{11.3}$$

However, this idea also has a problem, which is that the node only has one chance to form a block, unlike in PoW where the node can try different nonces until it finds a valid one. This makes the PoS mechanism less flexible.

For a fixed T, there is always a non-zero probability that no one can satisfy this equation; in this case, the protocol simply stalls forever.

11.2.2 Idea 3

We can give everyone another chance if the first round of the lottery does not have a winner. We can use one more factor in the hash function: the **timestamp** (ts).
We can divide the time into small units, for example, each node makes one attempt every second.

Definition 11.2.3: POS Idea 3

A node n succeeds in mining a block if

$$H(\text{hash(parent.header)}, ts, pk_n) < T \times stake_n \tag{11.4}$$

However, there are two problems with this idea, which are related to the security and fairness of the PoS mechanism.

- the block content is not tamper-resistant against an adaptive adversary, who can change or add transactions after winning the lottery.

- the public election is vulnerable to bribery or corruption, as well as not resistant to an adaptive adversary, who can influence or attack the block proposers.

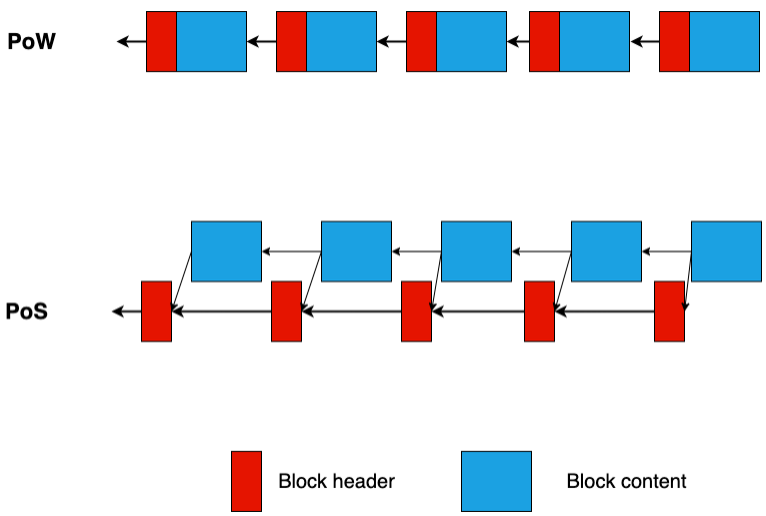


Figure 11.1: PoW vs. PoS

The block headers are connected by hashes, but the block contents also have hashes that link them together (see Figure 11.1). This way, the attacker cannot change the whole chain if some blocks belong to honest nodes. But, an adaptive attacker can still take over all honest nodes that have blocks on the chain and then change any block they want. We can stop this by using a special technique called **Key Evolving Signature (KES)**.

11.3 Key Evolving Signature (KES)

Key-Evolving Signatures (KES) is used to sign blocks in a PoS mechanism. KES allows the block proposer to update their secret key in every step, while keeping their public key the same. The old secret key is erased, so that it cannot be used to sign old blocks. This prevents an attacker from forging old signatures with new keys, and helps achieve adaptive security. Adaptive security means that the PoS mechanism can resist an attacker who tries to corrupt old blocks sometime in the future.

A simple example of KES involves periodically changing the secret key of any coin by its hash output. The block content has to be signed by the lottery winner with KES. We can make honest nodes delete all the old keys, which makes sure that the content of honest blocks cannot be changed by an adaptive attacker.

Verifiable Random Function (VRF)

The public keys of the nodes are known to everyone, so the lottery winner is not a secret. This means that an attacker can try to harm or bribe a node that will win the lottery in the future. We can prevent this by using a special technique called **verifiable random function (VRF)**

VRF is a function that takes a secret key and an input, and produces an output and a proof. The output is random and unpredictable, but anyone can verify that it was generated by the secret key and the input, using the public key and the proof.

Theorem 11.3.1 VRF

$$VRF(sk, x) \rightarrow (y, \pi) \tag{11.5}$$

The secret key sk and the input x are used to generate the output y and the proof π

$$Verify(pk, x, y, \pi) \rightarrow True/False \tag{11.6}$$

The public key pk, the input x, the output y, and the proof π are used to check if the output is valid or not.

Definition 11.3.1: POS idea 4

A node n succeeds in mining a block if

$$VRF(hash(parent.header), ts, sk_n) < T \times stake_n \tag{11.7}$$

VRF makes the block proposer selection more random and secret, so that the attacker cannot predict or influence who will win the lottery. VRF is therefore a powerful tool for enhancing the security and fairness of the PoS mechanism.

11.4 Nothing at Stake attack

The honest participants follow the longest chain rule, which means they only mine on the end of the longest chain. The attacker, however, mines on many places in the chain tree, and tries to create a longer chain than the honest one. The attacker can do this because they can use the same stake to mine many blocks without spending much resources.

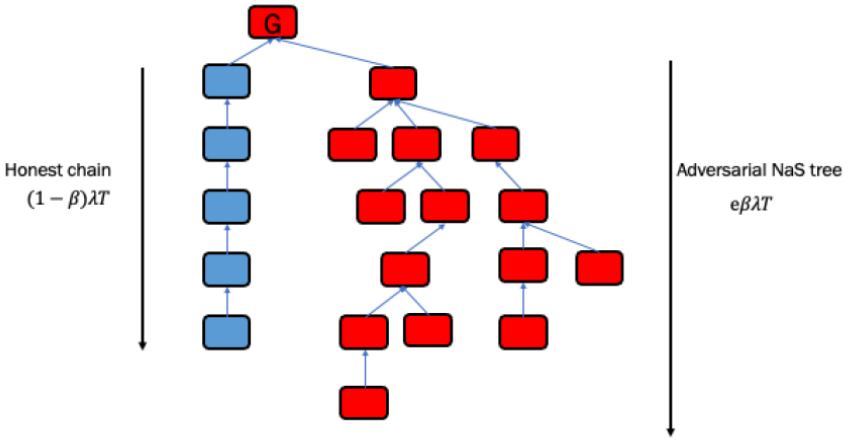


Figure 11.2: NaS attack

The honest chain grows linearly in time, like a Poisson process, with a rate of $(1-\beta)\lambda T$, while the attacker's chain grows exponentially in time, as a NaS tree, with a length of $e\beta\lambda T$, where e is the base of natural logarithm. NaS attack allows the attacker to compete with the honest participants unevenly, as they can cheat and manipulate the chain. The effect of the NaS attack is to amplify the growth rate of the private chain by a factor of e.

The NaS attack is successful when

$$(1-\beta)\lambda T > e\beta\lambda T \quad \Rightarrow \quad \beta < \frac{1}{1+e} \cong 27\% \tag{11.8}$$

We have seen a balancing attack on GHOST, which can work with less than 50% of the hash power. Can we do something similar with PoS? The answer is no, because the private attack is the worst case scenario for PoS. The protocol can resist any attack as long as $\beta < \frac{1}{1+e}$. This means that the attacker's stake must be less than 27% of the total stake. This is the exact security threshold for this protocol. It is about half of the security threshold for PoW, which is 50% of the hash power.

11.5 Boosting Security Threshold

We might wonder if we can make the threshold 50%, like in PoW longest chain. We want to do this with a simple change in the PoS protocol. We notice that the previous PoS protocol is weak to the NaS attack, because each block in the chain tree gives a new random source and starts a new lottery. We can fix this by changing the random source less often.

Definition 11.5.1: Ouroboros PoS

A first order idea is to never update the randomness and only draw it from the genesis block.

The Ouroboros PoS protocol proceeds in discrete epochs; essentially a new genesis block starts off each epoch.

Definition 11.5.2: POS idea 5

A node n succeeds in mining a block if

$$VRF(hash(Genesis), ts, sk_n) < T \times stake_n \tag{11.9}$$

Some recent studies have shown that the protocol in Equation 6 is safe if most of the stake is honest, which is the same as the PoW longest chain protocol. They also showed that the private attack is the hardest attack to stop, and the security threshold (on honest stake) is the best one. They used the same proof method of Nakamoto blocks that we used for the NaS attack.

The adversary can also try to bribe some of the block proposers in the Ouroboros Praos protocol by setting up a website that offers rewards for their credentials in a future epoch (see Figure 11.3). This way, the adversary can learn who will be the next block proposers before anyone else (because of VRF) and use them to launch a deadly attack.
For example, the adversary can create a forked version of the blockchain with the help of the bribed proposers and keep it secret until a block s is confirmed by the k-deep rule. Then, the adversary can reveal the forked blockchain to everyone, making it the longest chain and replacing s with s (which contains a double spend). Note that this attack only requires $k+1$ out of $2k+1$ lotteries to be won by the bribed proposers, who may have a very small stake in total.

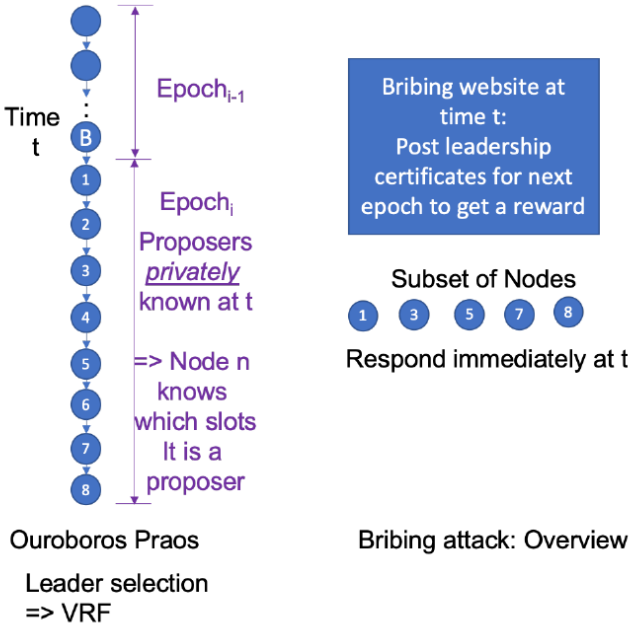


Figure 11.3: Structure of bribing attack

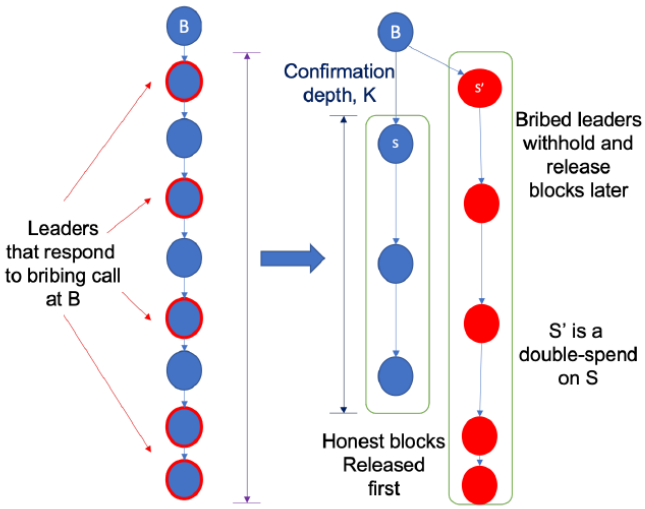


Figure 11.4: Bribing attack on Ouroboros Praos

11.5.1 c-correlation PoS protocol

The c-correlation property states that the updates of the randomness source are correlated with the multiples of a parameter c, which controls the security level of the protocol. In this rule, the common source of randomness remains the same for c blocks, and is updated only when the current block to be generated is at a depth that is a multiple of c.

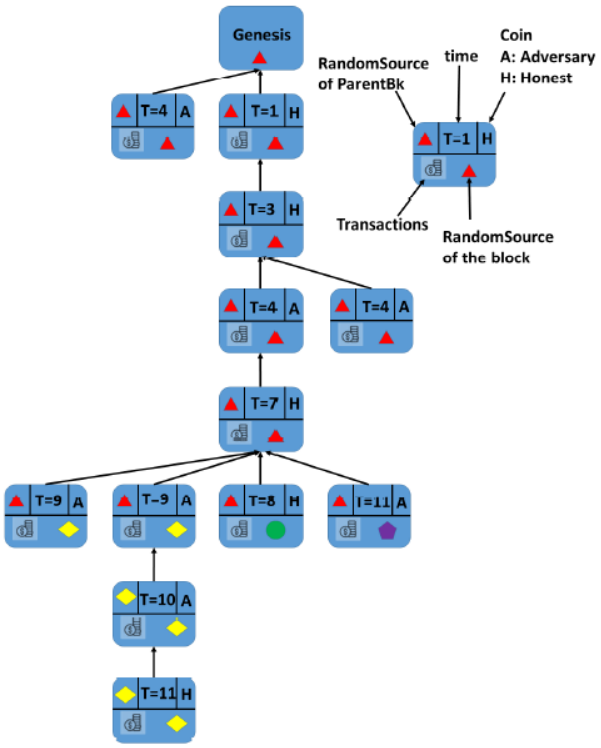


Figure 11.5: A snapshot of a block tree under c-correlation protocol with $c = 5$.

When updating, the hash of block header is used as the new source of randomness, i.e.,

$$\text{RandSource}(b) := \begin{cases} \text{VRF}(\text{RandSource}(\text{parent}(b)), ts, sk) & \text{if depth}(b)\%c = 0 \\ \text{RandSource}(\text{parent}(b)) & O.W. \end{cases} \tag{11.10}$$

Definition 11.5.3: POS idea 5.2

A node n succeeds in mining a block if

$$VRF(\text{RandSource}(Genesis), ts, sk_n) < T \times stake_n \tag{11.11}$$

The protocol in Equation (11.7) has the lowest value of c, which is 1. This means that the randomness source updates with every block, but it also makes the NaS attack very easy. The protocol in Equation (**??**) has the highest value of c, which is ∞. This means that the randomness source updates only once per epoch, but it also makes the NaS attack impossible. By changing the value of c, we can adjust the security level of the protocol (see Table 1). Figure 6 shows that with c-correlation, we can achieve the same security as the current Ouroboros protocol used by Cardano, but with a much smaller epoch size.

c	1	2	3	4	5	6	7	8	9	10
ϕ_c	e	2.22547	2.01030	1.88255	1.79545	1.73110	1.68103	1.64060	1.60705	1.57860
β_c^*	$\frac{1}{1+e}$	0.31003	0.33219	0.34691	0.35772	0.36615	0.37299	0.37870	0.38358	0.38780

Table 1: Numerically computed growth rate ϕ_c and stake threshold β_c^*.

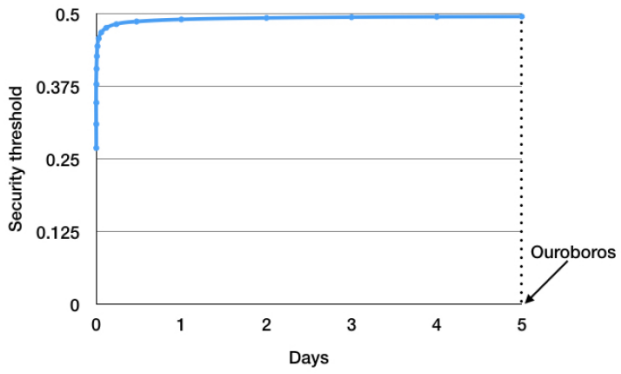


Figure 11.6: The security threshold β_c^* c of c-correlation against the epoch size, equaling to c times the inter-block time, which we set to be 20s, to match the implementation of Ouroboros in Cardano. The Cardano project currently updates the common randomness every 5 days (21600 blocks), while the security threshold of c-correlation can approach 1/2 with much higher randomness update frequency

11.6 Dynamic Stake

So far the stake of the nodes that determines their success in the lottery is fixed, i.e., not changing.
This is bad for two reasons:

- A node that has no stake now can still participate in the lottery (making it hard for new nodes to join)
- A node that has a lot of stake now cannot use its coins for anything else (making it easy for old nodes to dominate)

A better way is to let the dynamic stake of the nodes decide their success in the lottery; after all, the ledger is updated dynamically as new blocks are added to the longest chain and confirmed (when deep enough) and the security of the PoS longest chain rule ensures that all the honest nodes agree on the ledger.

The stake of a node n in the dynamic stake setting depends on both time and chain. Different chains in the blocktree have different transactions, which affect the stake of each node. We need to say which chain we are talking about, when we look at the stake of a node. If we just use the state of the previous block to decide the stake of each node, the adversary can grind on the stake of its coin.

The stake grinding attack is when the adversary uses its stake to increase its chances of winning the next round. The adversary does this by creating two blocks with the same header, but different transactions. In one block, it moves its stake from one coin to another; in the other, it does the opposite. In the example of Figure 7, the adversary has coin A and coin B. When it can propose a block, it makes two blocks: one with A to B, and one with B to A. In the next round, the adversary can use both coins to play the lottery, so it has twice the chance of winning by changing its stake.

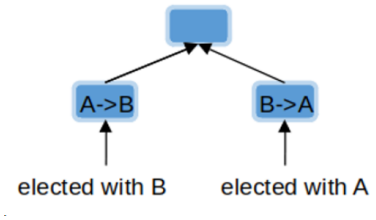


Figure 11.7: Stake grinding attack

A simple way to stop the stake grinding attack is to use the stake in the block that is s blocks behind the current block on the main chain, where s is big enough so that everyone agrees on that block. But this still has a problem. Imagine the adversary makes its own secret chain from the start (or any block in the blocktree). At first, the secret chain grows at a speed of $\beta\lambda$. But after s blocks from the start of the secret chain, the adversary can start changing the private key of its coin; when it finds a good coin, it can move its stake to that coin by adding transactions in the first block in the secret chain. This is possible because all the blocks in the secret chain are made by the adversary. It can change anything in the secret chain and sign all the previous blocks again.

The adversary can use this stake grinding attack to win the lottery every time in the secret chain, and eventually catch up with the public blocktree, which grows at a fixed speed $(1-\beta)\lambda$. To stop this attack, we suggest a new way to choose between forks called s-truncated longest chain.

11.6.1 New fork choice rule

The stake grinding attack makes the private chain grow slowly at first, so we cannot just compare the length of the chains when they split. We need a different way to choose the best chain. We use the s-truncation rule, which compares the density of the chains after they fork. The density is how many blocks are in the first s blocks after the fork. The honest node always has one main chain that it adds its new block to. When it gets a new chain of blocks, it has to decide which chain to keep. It does not look at the length of the two chains, like in the longest chain rule. It looks at the time it took to make the first s blocks after the fork in both chains. The chain that is faster and denser is the better chain.

The block where the two chains diverge is b_{fork}. The honest node compares how much time each chain takes to produce s blocks after b_{fork}. The chain that is faster and denser is the better chain. The honest node will attach its new block to the last block of that chain.

We have one more thing to say about the s-truncation rule. We only use it when we compare two chains that both have s or more blocks after they split. If one of the chains has less than s blocks after the split, we use the longest chain rule to choose the best chain. This is how we make the PoS lottery work well, even when the stake changes over time.

11.7 Dynamic Availability Using VDF

Bitcoin has a nice feature called dynamic availability: Bitcoin can deal with a changing and unknown level of mining power; Miners can come and go as they wish without any sign-up process. Is it possible to have the same feature in the PoS version?

The protocol we have shown so far works well if only a small part (for example, 27% for c = 1) of the online nodes are adversarial, but it also needs another thing: all the bad nodes are always online. This might seem fair (why would a bad node go offline?), but in real life this can be very hard. Usually, in public blockchains, PoW or PoS, no node is bad at the start of a new blockchain token (because the token is not worth much); bad nodes only show up later on.

The protocol needs another thing to work well: all the bad nodes are always online. This is not just extra, but very important for the security. Imagine that in the first year of the PoS blockchain, only 10% of the stake is online. All of them are good nodes. Then, in the second year, all 100% of the stake is online, but 20% is bad. Even though most of the online stake is good, the protocol is not safe because the bad nodes can use their 20% stake to go back in time and win all the past blocks from the start. Then they can make a chain very fast from the start and make it longer than the current chain (see Figure 11.8(a)). This is possible because making blocks in the PoS protocol is very cheap (just need to play all the old lotteries). So, when bad nodes come online, they can change not only the present, but also the past. PoW does not have this problem because it takes a lot of time and work to make a chain from the past and that chain will always be shorter than the current chain. So, PoW has a sense of time, which means nodes cannot change the past blocks when they were offline. This is why PoW can handle dynamic nodes, where both good and bad nodes can come and go (see Figure **??**(b)).

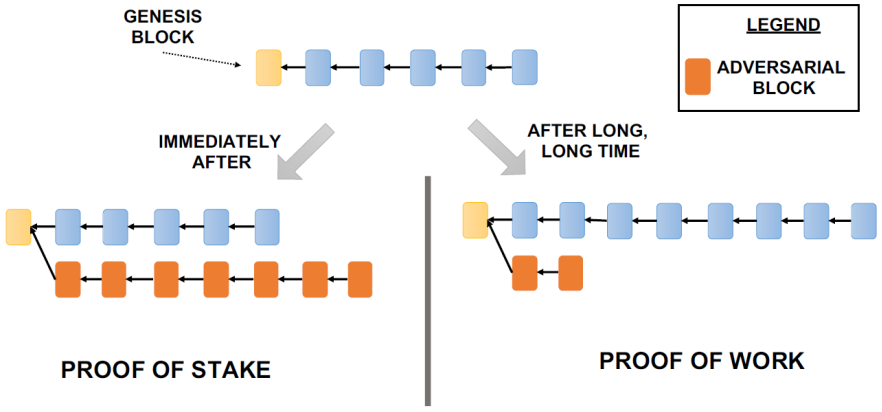


Figure 11.8: (a) Newly coming online stakeholders in the PoS protocol can grow a chain from genesis instantaneously. (b) Newly joined miners in the PoW protocol take a long time to grow such a chain and is always behind.

11.7.1 PoSAT

PoSAT is a new idea that solves the problem of dynamic nodes in PoS. It uses a special kind of function called **Verifiable Delay Function (VDF)** to make the block lottery depend on time. VDFs are built on top of iteratively sequential functions, i.e., functions that are only computable sequentially: $f^l(x) = f \circ f \circ ... \circ f(x)/$, along with the ability to provide a short and easily verifiable proof that the computed output is correct.

Examples of such functions include (repeated) squaring in a finite group of unknown order, i.e., $f(x) = 2^x$ and (repeated) application of secure hash function (SHA-256), i.e., $f(x) = Hash(x)$.
The VDF also gives a proof for each answer, and the proof can be checked quickly. This is amazing! VDFs can show how much time has passed (if we know how fast the CPU is), and they can also make random numbers. So, we can use VDF to run the lottery until a random time L when $f^L(x)$ is small enough. Then L will be a random number with a certain shape. This VDF lottery can create a sense of time in PoS.

Definition 11.1: PoS idea 6

A node n succeeds in mining a block if

$$VDF(\text{RandSource}(parent), ts, sk_n) < T \times stake_n \tag{11.12}$$

With VDF, the adversary cannot go back in time and create a longer chain than the honest chain immediately, because creating a longer chain now requires time. While the adversarial chain is growing, the honest chain is growing at a larger speed. Therefore, PoSAT is secure in the dynamic available setting.

Chapter 12

Layer 2 Scaling: Side Blockchains

12.1 Introduction

In the recent series of lectures, we have delved into various strategies to enhance the scalability of blockchain systems, addressing aspects like throughput, latency, storage, communication, computation, and energy consumption. In these proposals, a common thread has been the modification of the longest chain consensus protocol. However, making changes to the consensus layer of a practical, operational blockchain system presents significant challenges. Such changes necessitate achieving a "meta consensus" among the participating nodes, essentially agreeing on how to alter the consensus mechanism itself. Even if a successful transition to a new consensus mechanism is achieved, it typically results in a "hard fork" in the ledger, where the blockchain splits into two branches following different consensus protocols.

Given this context, the focus of this lecture shifts towards proposals that aim to enhance performance without necessitating changes to the core consensus layer. These proposals operate at what is commonly referred to as "layer 2," where performance scaling is achieved by offloading computation and storage. Importantly, these mechanisms do not compromise the underlying security or integrity of the blockchain system at the "layer 1" level.

Among these layer 2 proposals, we'll explore two particularly promising approaches in this lecture: side chains and their ability to support a general account-based model and smart contracts. Side chains offer a means to achieve enhanced performance while maintaining compatibility with the fundamental principles of blockchain technology. This exploration will shed light on how these mechanisms work, their benefits, and how they contribute to the overall scalability and efficiency of blockchain systems.

12.2 Side Blockchains

A side blockchain is essentially a smaller-scale blockchain that represents a subset of trust, often characterized by a reduced number of participating nodes or hash power. This side blockchain is interconnected with a trusted main blockchain through a mechanism where the nodes of the side blockchain periodically commit the cryptographic hashes of their blocks to the main blockchain (as depicted in Figure 1a). The ordering of blocks within the side blockchain is determined by the sequence of these committed hashes on the trusted main blockchain. This interconnected structure ensures that the security of the side blockchain is directly derived from the underlying security of the trusted main blockchain.

This approach offers a straightforward, practical, and efficient solution. It allows a single trusted main blockchain to efficiently accommodate numerous side blockchains. Unlike a full-fledged blockchain, the trusted main blockchain is not burdened with the need to store, process, or validate the content or semantics of the blocks on the side blockchains. Its role is primarily focused on maintaining and sequencing the hashes associated with these blocks. As a result, this setup enables side blockchains to remain secure and operational even if they lack an honest majority of participants.

From a broader perspective, this architecture shares similarities with the concept of a uni-consensus-based sharded blockchain, which was explored in the previous lecture. Both approaches leverage the concept of interconnected blockchains to achieve enhanced scalability and performance, while maintaining security through their connection to a trusted main blockchain.

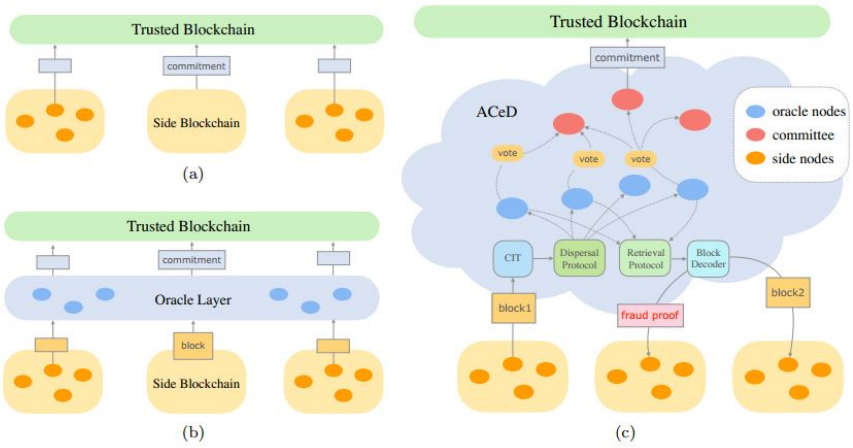


Figure 12.1: (a) Side blockchains commit the hashes of blocks to a larger trusted blockchain. (b) An oracle layer is introduced to ensure data availability. (c) ACeD is a scalable data availability oracle.

12.2.1 Data Availability Attack

The *data availability attack* is a significant vulnerability that affects both the side-blockchain scheme and the sharded blockchain scheme. In this attack, a malicious node within a side blockchain network (or a shard) commits the hash of a block to the trusted main blockchain without actually transmitting the block's data to the other nodes. This creates a dilemma for users who are trying to construct a coherent ledger from the blockchain: should they wait until they receive the missing block's data, potentially leading to a loss of liveness, or should they proceed without the block's data, risking potential safety violations?

The data availability attack was initially introduced in the context of *light clients* in blockchains, as highlighted by Vitalik Buterin of Ethereum in a note and an accompanying paper. Light nodes store only block headers and verify proof-of-work criteria, relying on full nodes to validate blocks and provide *fraud proofs* for invalid blocks.

In the case of light nodes, the data availability attack is not as fatal. Miners, who often run full nodes, can choose to ignore unavailable blocks and mine on their parent blocks. Honest miners will naturally avoid building on unavailable blocks, causing them to fall out of the longest chain over time, after which light nodes can safely ignore them.

However, the attack becomes more serious when applied to side blockchains and sharding. The key difference is that a side blockchain (or shard) might lack an honest majority of miners or full nodes. In such scenarios, there is no uniform way for participants within a side blockchain (or shard) to decide whether to include a missing block in their ledger. To address this challenge, a data availability oracle is proposed as a solution. This oracle helps nodes in the system determine whether a specific block's data is available, while defending against various forms of data availability attacks.

The properties that a data availability oracle must satisfy include the ability to provide nodes with information about the availability of block data, even in the face of adversarial attacks. It must offer a robust security guarantee, especially in the context of side blockchains and sharding, where the attack's impact can be more severe. Early techniques designed to mitigate the data availability attack, particularly those aimed at protecting light nodes, [1, 4] may not provide as strong of a security guarantee as a dedicated oracle.

A comparison of the data availability attack and its solutions for light nodes versus side blockchains is summarized in Table 1, highlighting the distinct challenges and considerations in each scenario.

12.2.2 Data Availability Oracle

A data availability oracle serves as an intermediary layer that interacts with side blockchains, ensuring the availability of data to these side blockchains while maintaining a connection with the trusted main blockchain. This oracle layer plays a vital role in verifying and committing blocks from side blockchains to the main blockchain.

Table 12.1: Data availability attack in two scenarios

Bitcoin light nodes	Sidechain clients
Light nodes random sample chunks	Oracle nodes store dispersed data
Rely on one honest full node to reconstruct the block	Any client can reconstruct the block
Probabilistic secure: need enough light nodes to ensure reconstruction	Deterministic secure: specifc protocol to guarantee reconstruction

The process involves a consensus mechanism among the oracle nodes to determine whether a proposed block is retrievable (i.e., its data is available), and only upon reaching a consensus is the block committed to the trusted blockchain (as depicted in Figure 1b).
There are two primary approaches to constructing such an oracle:

1. **Repetition:** In this approach, each oracle node maintains a complete copy of the block. The oracle nodes engage in a voting process to decide whether the block's data is available. While this method is straightforward and ensures reliability through redundancy, it comes with communication and storage overhead that scales with the number of oracle nodes. This approach is feasible when the number of oracle nodes is small and considered trustworthy. However, in a fully decentralized context, scalability becomes an issue.

2. **Dispersal:** This approach involves breaking the block into multiple chunks and distributing these chunks evenly among the oracle nodes. Each oracle node only holds a portion of the block's data. Dispersal minimizes data redundancy and storage overhead compared to the repetition approach. However, it introduces a security challenge. If even a single malicious oracle node is present, it can compromise the retrievability of the entire block. The trade-off between security and scalability is evident here.

The key challenge lies in finding an optimal balance between security and scalability while efficiently sharing block data among the oracle nodes to ensure data availability. This challenge is particularly important in scenarios where both trust and performance are critical considerations. The choice between repetition and dispersal depends on factors such as the number of oracle nodes, their trustworthiness, and the specific requirements of the blockchain ecosystem.
Ultimately, a well-designed data availability oracle provides a crucial mechanism for enhancing the security and reliability of side blockchains while maintaining a strong connection to the trusted main blockchain. It enables the scaling of performance and throughput without compromising on the integrity of the system.

12.2.3 Erasure Coding

Erasure coding is introduced as a solution to enhance the scalability and reliability of data availability oracles. It addresses the challenge of ensuring data availability while minimizing the impact of malicious nodes hiding or corrupting data. By applying erasure codes to the data chunks of a block, redundancy is added, allowing for the recovery of the entire block even when a fraction of the data chunks is missing or compromised.
The dispersal protocol, as mentioned earlier, faces a vulnerability when a single malicious node hides a data chunk, rendering the entire block unrecoverable. Erasure coding mitigates this vulnerability by introducing redundancy through coding techniques. With erasure coding[2], only a fraction of the data chunks needs to be available to reconstruct the full block, making the system more resilient to malicious actions. This fraction is determined by the undecodable ratio , which is influenced by the specific erasure code and decoding algorithm used.
For instance, consider the use of a (n, k) Reed-Solomon (1D-RS) code[3], a well-known erasure code. In this scenario, a block B is divided into k data symbols, each of size $\frac{b}{k}$ bytes. These data symbols are then linearly combined to generate a coded block C, consisting of n coded symbols. If the undecodable ratio is set to $\frac{1}{3}$, and there are n oracle nodes, each assigned one coded symbol to store, then any subset of $\frac{2}{3}$ coded symbols within C is sufficient to reconstruct the entire coded block.
This approach significantly enhances the security of the data availability oracle, allowing for a portion of the data to be missing or tampered with while still ensuring the ability to recover the complete information. Erasure coding introduces redundancy in a way that aligns with the characteristics of the chosen erasure code and decoding algorithm, providing a practical and efficient solution to the data availability problem in the context of blockchain side chains.

12.2.4 Coding integrity and correctness

In the context of ensuring coding integrity and correctness in a data availability oracle, there are challenges posed by potential attacks such as the incorrect-coding attack. This attack involves a malicious block producer constructing a Merkle tree with nonsensical symbols in such a way that it becomes impossible to successfully reconstruct the original block. To counter this, an incorrect-coding proof or fraud proof mechanism is employed, which allows nodes attempting to reconstruct the data to detect symbols that fail the parity check.

Table 12.2: Performance metrics for different data availability oracles (N: number of oracle nodes, b: block size)

	maximal adversary fraction	normal storage overhead	normal download overhead	worst storage overhead	worst download overhead	communication complexity
uncoded (repetition)	$\frac{1}{2}$	$O(N)$	$O(1)$	$O(N)$	$O(1)$	$O(Nb)$
uncoded (dispersal)	$\frac{1}{N}$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(b)$
1D-RS	$\frac{1}{2}$	$O(1)$	$O(1)$	$O(b)$	$O(b)$	$O(b)$
2D-RS [1]	$\frac{1}{2}$	$O(1)$	$O(1)$	$O(\sqrt{b}logb)$	$O(\sqrt{b}logb)$	$O(b)$
ACeD	$\frac{1}{2}$	$O(1)$	$O(1)$	$O(logb)$	$O(logb)$	$O(b)$

Table 12.3: System Performance Metrics

Metric	**Formula**	**Explanation**
Maximal adversary fraction	β	The maximum number of adversaries is $\beta * N$.
Storage overhead	$\frac{D_{store}}{D_{info}}$	The ratio of total storage used and total information stored.
Download overhead	$\frac{D_{download}}{D_{data}}$	The ratio of the size of downloaded data and the size of reconstructed data.
Communication complexity	D_{msg}	Total number of bits communicated.

The efficiency of the fraud proof method is influenced by the size of the proof. In the case of 1D Reed-Solomon (1D-RS) coding, the fraud proof contains k coded symbols, which is not significantly better than simply downloading the original block consisting of n symbols. To address this limitation and reduce the size of the fraud proof, 2D Reed-Solomon (2D-RS) coding has been proposed. In 2D-RS [1], a block is organized into a matrix of size $(\sqrt{k}, \sqrt{k})$, and both rows and columns are encoded using Reed-Solomon codes to generate n^2 coded symbols. This technique reduces the size of the fraud proof to $O(\sqrt{b}logb)$, assuming a constant symbol size. Another approach to reducing the proof size is the use of a cryptographic hash accumulator called Coded Merkle Tree (CMT)[4]. The CMT method further reduces the proof size to $O(logb)$, making it more efficient in terms of data availability checks. However, it's important to note that CMT is designed for light nodes to randomly sample coded symbols for data availability verification, providing probabilistic security. As a result, it may not fully address the oracle problem for side blockchains, which requires a higher level of certainty and security.
In summary, ensuring the integrity and correctness of coding in the context of data availability oracles involves strategies such as fraud proofs, 2D Reed-Solomon coding, and cryptographic hash accumulators. These approaches aim to detect incorrect-coding attacks and provide efficient methods for verifying data availability while maintaining a high level of security.

12.3 AceD

Authenticated Coded Dispersal (ACeD) is a recent advancement aimed at addressing the data availability oracle

problem with a scalable solution. ACeD offers several improvements over other solutions, as highlighted in the performance comparison table (Table 2). The architecture of ACeD comprises four key components, depicted in Figure 1c:

1. **Coded Interleaving Tree (CIT):** This is a coded commitment generator that is constructed in a layered, interleaved manner while incorporating erasure codes. The interleaving property of CIT is designed to prevent the need for downloading additional proofs, thus minimizing the number of symbols required for storage.

2. **Dispersal and Retrieval Protocol:** ACeD incorporates a pair of protocols for dispersing and retrieving tree chunks across the network. These protocols are designed to optimize the distribution of data while ensuring high retrievability with minimal redundancy.

3. **Hash-Aware Peeling Decoder:** ACeD employs a hash-aware peeling decoder to achieve linear decoding complexity. This decoder is crucial for efficiently reconstructing the original data from coded symbols. Additionally, the use of a peeling decoder reduces the size of the fraud proof to a single parity equation, enhancing efficiency.

The ACeD approach represents a significant advancement in tackling the data availability oracle problem. By incorporating techniques such as the Coded Interleaving Tree, optimized dispersal and retrieval protocols, and a hash-aware peeling decoder, ACeD aims to provide a more scalable, efficient, and secure solution for ensuring data availability in side blockchains or similar architectures. The focus on minimizing redundancy and optimizing data distribution contributes to its enhanced performance compared to other approaches.

12.3.1 Coded Interleaving Tree

The Coded Interleaving Tree (CIT) offers efficiency similar to Coded Merkle Tree (CMT) in terms of fraud proof size, although they have distinct construction methods. CMT utilizes a pull model to randomly sample symbols through an anonymous network, making it suitable for light nodes only. Conversely, CIT is designed for secure deterministic dispersal with a push model, requiring no anonymity assumption, and being designed to be incentive compatible.
The construction of CIT is best understood through its stages. CIT takes a client-proposed block as input and produces three outputs: a commitment (CIT root), a sequence of coded symbols (leaves of CIT in the base layer), and their proof of membership (POM). The POM includes a Merkle proof (siblings' hashes from each layer) and a set of parity symbols from intermediate layers.
The construction of CIT is illustrated in Figure 2 using an example. Given a block of size b bytes and ℓ layers in CIT, the process begins by dividing the block into equal-sized chunks called data symbols, each with a size denoted as c. This results in $s\ell = \frac{b}{c}$ data symbols. Applying erasure codes with a coding ratio r (where $r1$) generates $m\ell = \frac{s\ell}{r}$ coded symbols in the base layer. By aggregating hashes of q coded symbols, $\frac{m\ell}{q}$ data symbols are created for the parent layer (layer $\ell - 1$), which can be further encoded into $m\ell - 1 = \frac{m\ell}{qr}$ coded symbols. This iterative aggregation and coding process continues until the number of symbols in a layer reduces to t, which is the size of the root.
Overall, CIT's construction combines erasure coding, hashing, and aggregation to create a tree structure that efficiently represents the original block's data in a secure and deterministic manner. This process enables CIT to achieve a fraud proof size similar to CMT while offering advantages in terms of its construction principles and applicability.

For all layers j except for the root, where $1 \leqslant j \leqslant \ell$, let's denote the set of all m_j coded symbols as M_j. This set can be divided into two disjoint subsets of symbols: data symbols, represented as S_j, and parity symbols, represented as P_j. The count of data symbols is $s_{\lfloor j \rfloor} = rm_j$, and they are specified as $Sj = [0, rm_j)$. On the other hand, the parity symbols are specified as $Pj = [rm_j, m_j)$. For a given block containing s data symbols in the base layer, the aggregation rule for the k-th data symbol in layer $j-1$ is defined as follows:

$$Q_{j-1}[k] = \{(H(M_j[x]))|x \in [0, M_j), k = x \mod rm_{j-1}\} \tag{12.1}$$

$$M_{j-1}[k] = H(\text{concat}(Q_{j-1}[k])) \tag{12.2}$$

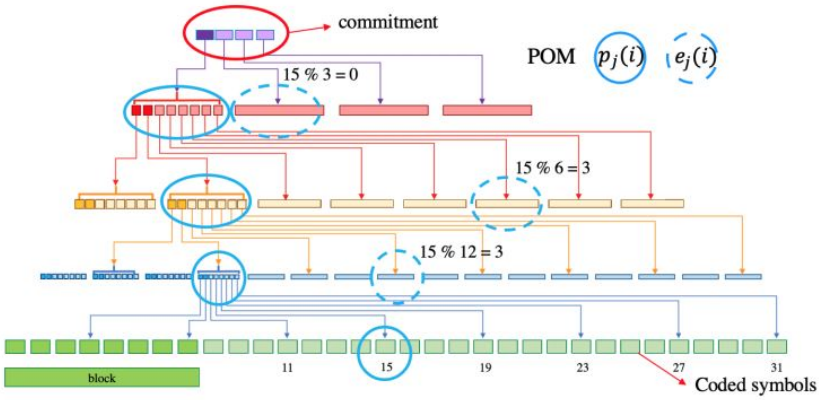


Figure 12.2: (1) CIT construction process of a block with $s\ell = 8$ data symbols, applied with erasure codes of coding ratio r = $\frac{1}{4}$. The batch size q = 8 and the number of hashes in root is t = 4. (2) Circled symbols constitute the 15th base layer coded symbol and its POM. The solidly circled symbols are the base layer coded symbol and its Merkle proof (intermediate data symbols), the symbols circled in dash are parity symbols sampled deterministically.

In the above equations, $1j$, and H denotes a hash function. The tuple Q[k] comprises hashes that will be utilized to generate the k-th symbol in the parent layer. Additionally, the function concat signifies the process of string concatenation, which involves combining all elements within an input tuple. This procedure remains consistent with the formulas provided earlier.

References

[1] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044, 2018*

[2] Shu Lin and Daniel J Costello. *Error control coding*, volume 2. Prentice hall, 2001.

[3] Irving S Reed and Gustave Solomon. Polynomial codes over certain fnite felds. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[4] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 114–134. Springer, 2020.

Chapter 13

Layer 2 Scaling: Payment Channels

Following the idea of previous chapter we aim to scale existing blockchain performance without changing the consensus layer. Given that the consensus protocol (layer 1) is untouched, these methods are known to afford Layer 2 scaling. Two prominent instances are side blockchains (can handle account based systems and smart contracts) and payment channels (focused on the UTXO state management system). In this chapter we will discuss about **payment channels**.

13.1 Payment Channels

Payment channels allow users to exchange transactions for multiple times off-chain and only broadcast the final state to the main blockchain. Users submit transactions on-chain to start the channel and handle a dispute. This reduces the load on the network and the fees for the users.
It locks parts of the state of the blockchain when starting a channel, processes transactions associated with this locked state in an application layer, and finally unlocks the channel with the updated state.

A UTXO system like Bitcoin consists of transactions that have inputs and outputs.Outputs serve as **locks** for a specific amount of funds, while inputs serve as keys of corresponding outputs to **unlock** the funds and transfer them between accounts.

Example 13.1.1 (Locking and unlocking script: Pubkeyhash)

A locking script is a condition that must be satisfied to spend an output. An unlocking script is a proof that the spender meets the condition.
Figure 13.1 illustrates an example of a transaction where Alice sends 1 BTC to Bob. The locking script requires that the spender provides a public key that matches the hash of Bob's public key, and a signature that proves the ownership of the private key. The unlocking script provides these two pieces of information. The nodes then execute the scripts and check if they are valid. If they are, the transaction is confirmed and Bob can spend the output.

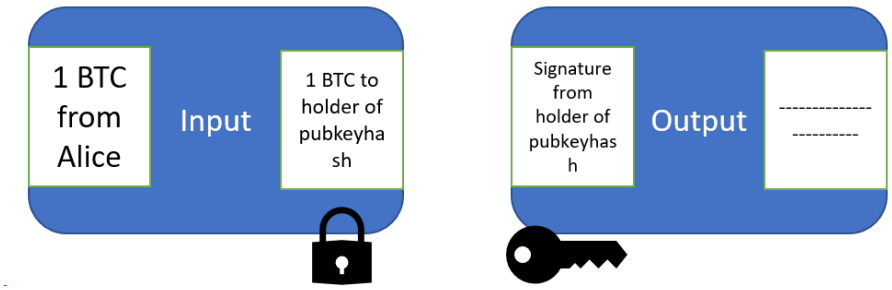


Figure 13.1: Example 13.1 Figure

There are three new types of cryptographic primitives that allow "flexible locks", so that the trust can be extracted outside the blockchain :

- **Multisig** : Locking transaction has to be signed by k out of n public keys.
- **Hashlock** : A hashlock is a type of cryptographic lock that can be opened by the owner of a public key (Bob) and a secret. The transaction has a locking script that contains the hash of the secret and a public key. The transaction can only be spent by Bob if he provides the secret and a signature that proves his ownership of the private key. The secret is a random value that is generated by Alice, who wants to send funds to Bob.
- **Timelock** : A timelock is a type of cryptographic lock that can be opened only after a certain time or block height.
  There are two kinds of timelocks:
  - **CLTV (Check Lock Time Verify)** : transaction can only be spent after a specific block height or timestamp.
  - **CSV (Check Sequence Verify)** : transaction can only be spent after a certain number of blocks have passed since the transaction was recorded in the blockchain.

13.2 One-Way Payment Channel

One-way payment channels only allow funds to flow in one direction, from payer to recipient. For example, suppose Alice wants to pay Bob in increments of 1 BTC up to a maximum of 10 BTC. the payment channel proceeds in the following steps:

1. **Creating the channel** :
   Alice signs a **funding transaction** and posts it on the blockchain to create the channel. The funding transaction has one input with 10 BTC that Alice signs. The output can either:
   - contain an address that is made from Alice and Bob's public keys (multiaddr = GetAddrbyAccount(pk1, pk2)). It needs both Alice and Bob's secret keys to unlock it. Neither of them can do it alone.
   - The output has Alice's address, and a **timelock** for the channel. If Bob is online, he can get the payment by posting a closing transaction. If not, the coins go back to Alice when the time is up.

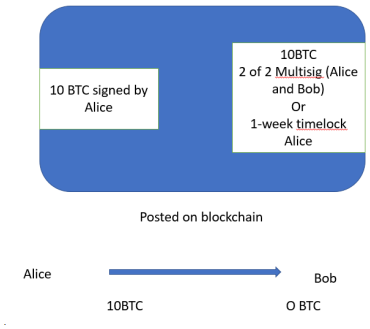


Figure 13.2: Funding transaction

2. **Updating the payments** :
   Each payment makes a new off-chain transaction, called a **commitment transaction**. A commitment transaction uses the same input as the funding transaction, and shows how the funds are split between Alice and Bob in the output, e.g. 9 BTC to Alice, 1 BTC to Bob.
   Alice can keep paying Bob like this in the channel. Every time she wants to send another 1 BTC to Bob, she makes a new commitment transaction from the same input and gives it to Bob.
   Bob keeps these transactions, and he can choose to post any one (but only one) of them to the blockchain to get the funds.

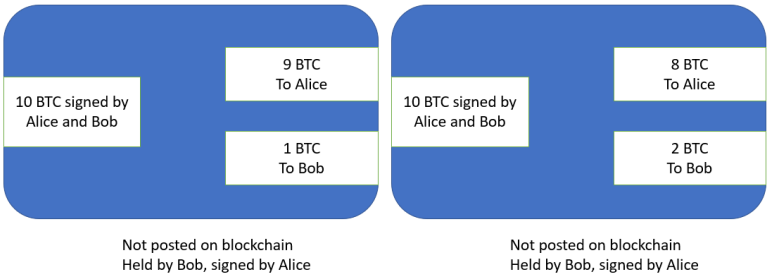


Figure 13.3: Commitment transaction

3. **Closing the channel** :
   To close the channel, a **closing transaction** is generated and posted on blockchain to close the channel and update the state. There are two possible ways to close the channel corresponding to the funding transaction:
   - **Cooperative**: Bob posts a commitment transaction that he and Alice both signed with their secret keys (2-of-2 multisig). The blockchain checks the signature and matches it with the address of the funding transaction. This confirms that Alice paid Bob.
   - **Non-cooperative**: Bob is offline and the channel runs out of time (e.g. one-day timelock). Alice posts the closing transaction on the blockchain. The transaction has Alice's signature and a time limit that has passed. Alice gets her money back

13.3 Two-Way Payment Channel

Two-way payment channels enable two parties to transfer funds to each other. Suppose now Alice and Bob both have 5 BTC to make payments. Consider the following steps :

1. **Creating the channel** :
   The funding transaction has two inputs, each with 0.5 BTC, one from Alice and one from Bob. The output has 1 BTC that needs both Alice and Bob's signatures (2-of-2 multisig). Before they start paying each other, Alice and Bob make secrets and share their hashes. The opening transaction will not be signed and posted on-chain until Alice and Bob receive a special commitment transaction respectively. The commitment transaction here will contain an input with 10 BTC signed by both Alice and Bob (the output of a funding transaction), and two outputs. For the transaction that is held by Alice, the outputs are (1) 5 BTC to Bob, and (2) 5 BTC with timelock to Alice or to Bob if he knows Alice's secret, these outputs are signed by Bob. Similar for Bob, the outputs are signed by Alice and contain (1) 5 BTC to Alice, and (2) 5 BTC with one-week timelock to Bob or to Alice if she knows Bob's secret.

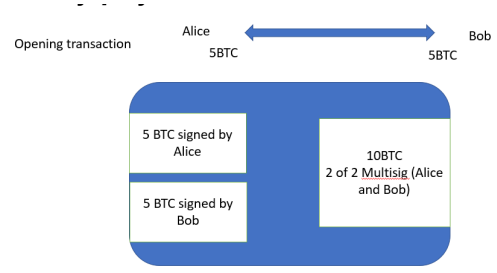


Figure 13.4: Two-Way payment channel

2. **Updating the payments** :
   Alice and Bob start generating a normal commitment transaction. Suppose Alice wants to send Bob 1 BTC, then she will sign a transaction with outputs: (1) 4 BTC to Alice, and 6 BTC with one-week timelock to Bob or to Alice if she knows Bob's secret. For each payment, Alice and Bob will generate a new secret and exchange the older secret to ensure that the older transaction will not be posted on chain

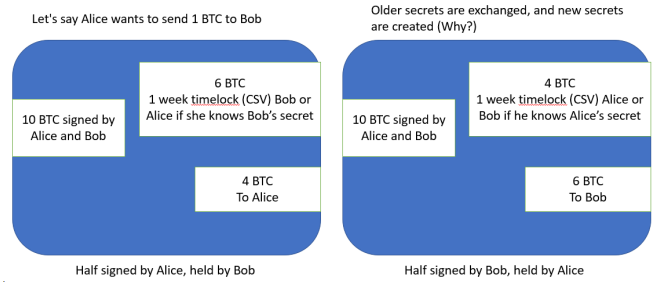


Figure 13.5: Commitment Transaction

3. **Closing the channel** :
   When one of the parties is offline, the channel can be closed by revealing any one of the commitment transactions in a non-cooperative way. And in the cooperative situation, Alice and Bob can create a transaction sending the settled balance to each party.

13.4 Multi-hop payment channels

Multi-hop payment is a technique that allows users to make payments across a network of payment channels without having to establish a direct channel between them.

13.4.1 Trusted multi-hop payments

Trusted multi-hop payments allow parties who do not have a direct channel to route their payments through intermediaries who do.

Example 13.4.1 (Trusted multi-hop payments)

Alice wants to pay Carol 1 BTC, but they do not have a channel between them. However, Alice and Bob have a channel, and Bob and Carol have a channel. Therefore, Alice can send 1 BTC to Bob, and Bob can forward 1 BTC to Carol, completing the payment. Alice and Carol have to trust Bob to forward the payment correctly and not steal the funds.

13.4.2 Trustless multi-hop payments using Hashlock

The secret is a random value that is generated by the receiver of the payment and shared with the sender through a secure channel. The sender then uses the hash of the secret to lock the payment in a transaction that can only be spent by the receiver if he provides the secret. This way, the sender can ensure that the payment will not be lost or stolen.

Example 13.4.2 (Trustless multi-hop payments using Hashlock)

Alice wants to pay Carol. Carol creates a secret S and sends its hash to Alice. Alice then sends 1 BTC to Bob using hashlock, with the condition that Bob can only claim it if he knows S. Bob then sends 1 BTC to Carol using hashlock, with the same condition. Carol can then reveal S to Bob and claim her payment. Bob can then use S to claim his payment from Alice. This way, Alice can pay Carol without trusting Bob, and Bob can act as an intermediary without risking his funds. See Figure 13.6

13.4.3 Trustless multi-hop payments using Hashlock and Timelock

Imagine the same scenario as of the Example 13.4.2. In addition to the previous transactions, now Carol also sends 1 BTC to Bob with a timelock of 1 week and also Bob sends 1 BTC to Alice with a timelock of 2 weeks. Carol can claim the payment from Bob by revealing Secret, which also allows Bob to claim the payment from

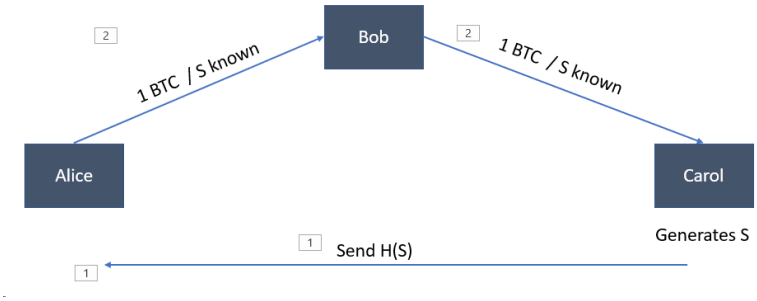


Figure 13.6: Trustless multi-hop payments using Hashlock

Alice. If Carol does not reveal Secret before a week, Bob gets his 1 BTC back. If Bob does not reveal S before 2 week, Alice gets her 1 BTC back. This way, the payment is trustless and secure.

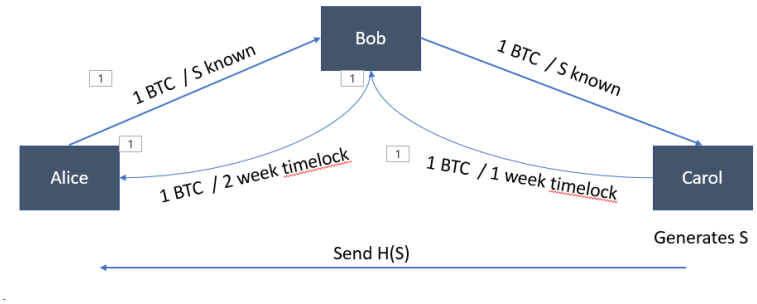


Figure 13.7: Trustless multi-hop payments using Hashlock and Timelock

13.5 Payment Network

A payment network is a system that allows users to transfer money or other assets across a network of participants. One of the challenges of designing a payment network is to enable multi hop payments, which are payments that involve more than two parties and require routing through intermediate nodes.

A lightening network is a solution for bitcoin that uses multi hop payment channels and routing to enable fast, cheap, and scalable transactions. A lightening network is a network of payment channels that are connected by nodes that can route payments between different channels.

The fee structure of the Lightning Network is based on two components:

- **Base fee** : The base fee is a fixed amount of Satoshi that is charged for each payment that is forwarded through a node.
- **Fee rate** : The fee rate is a variable amount of Satoshi that is charged for each million Satoshi that is transferred.

The total fees are calculated as follows:

$$\text{Amount} \times \text{Fee rate} + \text{Base fee}$$

The Lightning Network fees are much lower than the on-chain fees and can enable microtransactions that are otherwise not feasible on the blockchain.

In a Lightning Network :

- The total number of participating nodes is around 20,000.
- The total number of channels is around 85,000.
- The total capacity is around USD 100 million or 5,000 BTC. The capacity is the amount of bitcoin that is locked in the payment channels and can be transferred through the network.
- The highest capacity node is a node that has the most bitcoin locked in its channels. The highest capacity node has around 650 BTC.

The benefits of Lightning Networks are listed below:

- High throughput
- Low latency
- Less fees

The list of drawbacks includes:

- **Routing**: The Lightning Network relies on nodes to route payments between different channels, which can pose some challenges such as:
  - **Traffic**: The network may experience congestion or delays due to high demand or low availability of nodes or channels.
  - **Balance based**: The network may not be able to route payments that exceed the available balance or capacity of the channels involved.
  - **Centralization** : The network may become more centralized as some nodes may have more influence or power than others due to their high capacity, connectivity, or reliability.
- **Nodes need to stay online** : The nodes that participate in the network need to be online and monitor their channels to ensure the security and validity of the payments. This may create a trade-off between convenience and security for the users. Alternatively, the users may choose to outsource the monitoring of their channels to third-party services called **watchtowers**, which can prevent channel theft or fraud. However, this may also introduce trust issues or privacy risks for the users.
- **Capital locked in channels**: The Lightning Network requires users to lock some amount of bitcoin in their channels to enable payments. This may reduce the liquidity or availability of bitcoin for other purposes, such as saving, investing, or spending on the blockchain.

Chapter 14

Layer 2 Scaling: Rollups

14.1 Introduction

Rollups are a scaling solution used by the Ethereum community to increase throughput on the Ethereum Mainnet by moving computation and state-storage off-chain1. Rollups "roll up" a bunch of transactions into one and come in two basic forms: optimistic rollups and zero-knowledge rollups.
Optimistic rollups make the assumption that all of the rolled-up data is valid, and that nobody is trying to fool the blockchain by hiding spurious transactions within rollups. To protect against fraudulent transactions, optimistic rollup protocols allow people to contest bunk trades. The fraudulent transaction is submitted directly on the Ethereum network to check if it's legit, and to settle the dispute.
Zero-knowledge rollups (also referred to as zk-rollups) work very differently. They rely on a piece of cryptography called a zero-knowledge proof, which allows someone to mathematically prove that a statement is true without disclosing additional information about that statement.
Rollups cut down on blockchain transaction costs by "rolling up" batches of transactions into a single one. They also speed things up: the rollup is very quick to perform and the Ethereum blockchain needs only to process a single transaction rather than many. That's useful when Ethereum maxes out at around 15 transactions per second unassisted, see Figure 1.

14.2 Transfer

In a Rollup transfer, anyone can publish compressed data on-chain in a batch. This batch contains the pre-state, post-state, and compressed data. The diagram shows a Rollup contract connected to a State root with a dotted line. The State root is connected to four nodes representing Alice, Bob, Charlie, and David with solid lines. The nodes representing Alice, Bob, and Charlie are connected with a solid line, while the node representing David is connected with a dotted line. The text on the nodes reads "Alice > Bob > Alice > Charlie > Bob > Charlie" and "David: compressed item. There is still enough data to determine how to update the state…". This illustrates

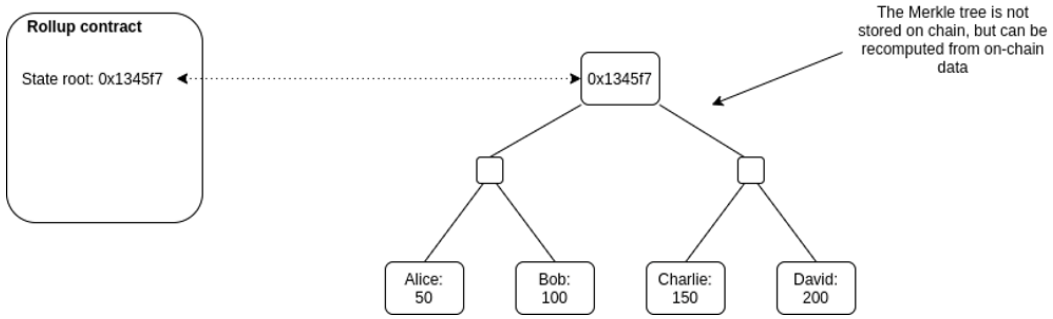


Figure 14.1: Rollups: A Scalable Solution for Ethereum - This diagram illustrates how Rollups execute transactions off-chain and report data on-chain in a compressed way, providing a scalable solution for Ethereum.

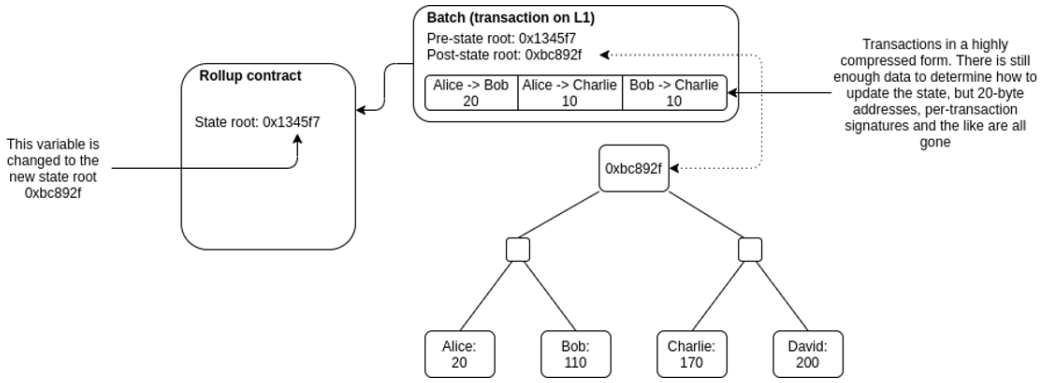


Figure 14.2: This diagram illustrates the concept of Rollups transfer in blockchain technology. It shows how anyone can publish compressed data on-chain, called a batch, which contains pre-state, post-state, and compressed data. The Rollup contract interacts with the state root, which in turn interacts with individual users such as Alice, Bob, and Charlie. The diagram also shows the balances of each user.

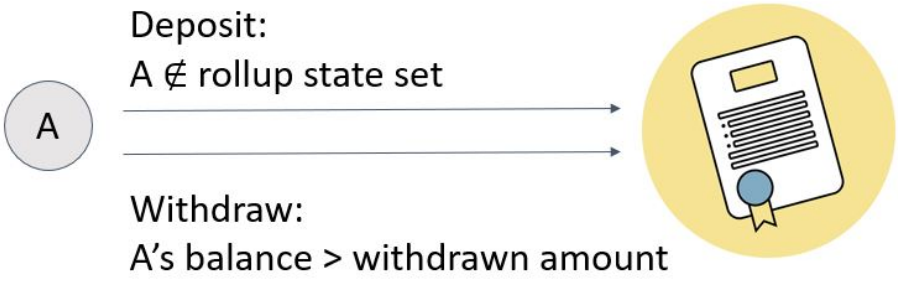


Figure 14.3: The image depicts a diagram that explains the concept of Rollups transfer in blockchain technology.

how Rollups can execute transactions off-chain while still maintaining the security and integrity of the blockchain, see Figure 2.

For more information see Figure 3, this diagram shows the process of publishing compressed data on-chain, referred to as a batch, which contains pre-state, post-state, and compressed data. The Rollup contract interacts with the state root, which in turn interacts with individual users such as Alice, Bob, and Charlie. The diagram also displays the balances of each user.

14.3 Optimistic Rollup

Optimistic Rollup is a Layer 2 scaling solution designed to enhance the scalability of blockchain networks like Ethereum. It operates on the principle that the majority of transactions are valid and honest. By assuming transaction validity upfront and using fraud proofs to handle exceptions, Optimistic Rollup significantly increases the throughput and reduces transaction fees on the mainchain.

14.3.1 Fraud Detection and Proofs

Optimistic Rollups are a type of Layer 2 scaling solution for Ethereum that involves aggregating multiple transactions into a single block and submitting it to the Ethereum blockchain. This allows for faster and cheaper

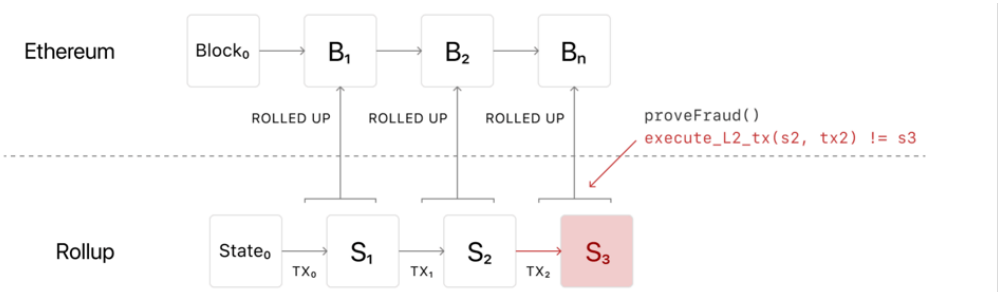


Figure 14.4: The blocks are labeled B1, B2, B3, etc., and the state transitions are labeled S1, S2, S3, etc. There is also a red arrow pointing to one of the blocks, indicating a fraud proof.

transactions.
Optimistic Rollups involve fraud proofs, which means that any user who sees an invalid state root can slash the aggregator by posting the valid state root. If a fraud proof is finalized, the chain is rolled back to the last valid state. This ensures the security and integrity of the system, see Figure 4.

Fraud proof generation and verification with Merkle tree
As we said before a Merkle tree is a data structure that allows for efficient verification of the contents of large data sets. It is a tree in which every leaf node is labeled with the cryptographic hash of a data block, and every non-leaf node is labeled with the cryptographic hash of the labels of its child nodes. In the context of fraud proofs, a Merkle tree can be used to efficiently verify the validity of a state root by comparing it to the computed post-state root, see Figure 5.

Complexity of fraud proofs Fraud proofs are essential components of Optimistic Rollup and similar Layer 2 solutions. They serve as mechanisms to detect and correct invalid or fraudulent transactions without requiring the complete re-execution of these transactions on the Layer 1 (L1) mainchain. However, the traditional approach to verify fraud proofs can lead to substantial gas costs and overhead. The complexity can be broken down as follows:

1. **Verify Fraud Proof Using a Verifier Contract:** The straightforward way to validate a fraud proof is by re-executing the disputed transactions on the Layer 1 chain. This involves interacting with a verifier contract deployed on the mainchain. The verifier contract checks the disputed transactions' correctness and enforces the correct state transition.

2. **Optimism and Arbitrum: Methods to Reduce Overhead:** Optimism and Arbitrum are two innovative approaches that aim to mitigate the overhead associated with fraud proofs while maintaining the security and scalability benefits of Layer 2 solutions.

   (a) **Optimism:** Optimism introduces an optimistic execution model, where transactions are initially assumed to be valid and executed on the Layer 2 chain, adhering to the optimistic assumption. This eliminates the immediate need for costly re-execution on the L1 mainchain. Users can submit transactions with minimal fees and faster confirmation times.
   In the event of a dispute, users or validators can present fraud proofs to the mainchain to challenge the optimistic assumption. However, the key innovation is that the system incentivizes disputers to provide the full transaction data for re-execution only if a dispute is valid. This introduces a "dispute fee" mechanism, where the successful disprover is rewarded from the funds of the disproven transaction. This approach encourages the submission of fraud proofs only when necessary, reducing the overall overhead.

   (b) **Arbitrum:** Arbitrum takes a similar approach to reduce overhead by utilizing a novel concept called "fraud proofs as data." Instead of requiring full re-execution on the mainchain, Arbitrum focuses on providing compact data that allows the mainchain to confirm the validity of transactions quickly and

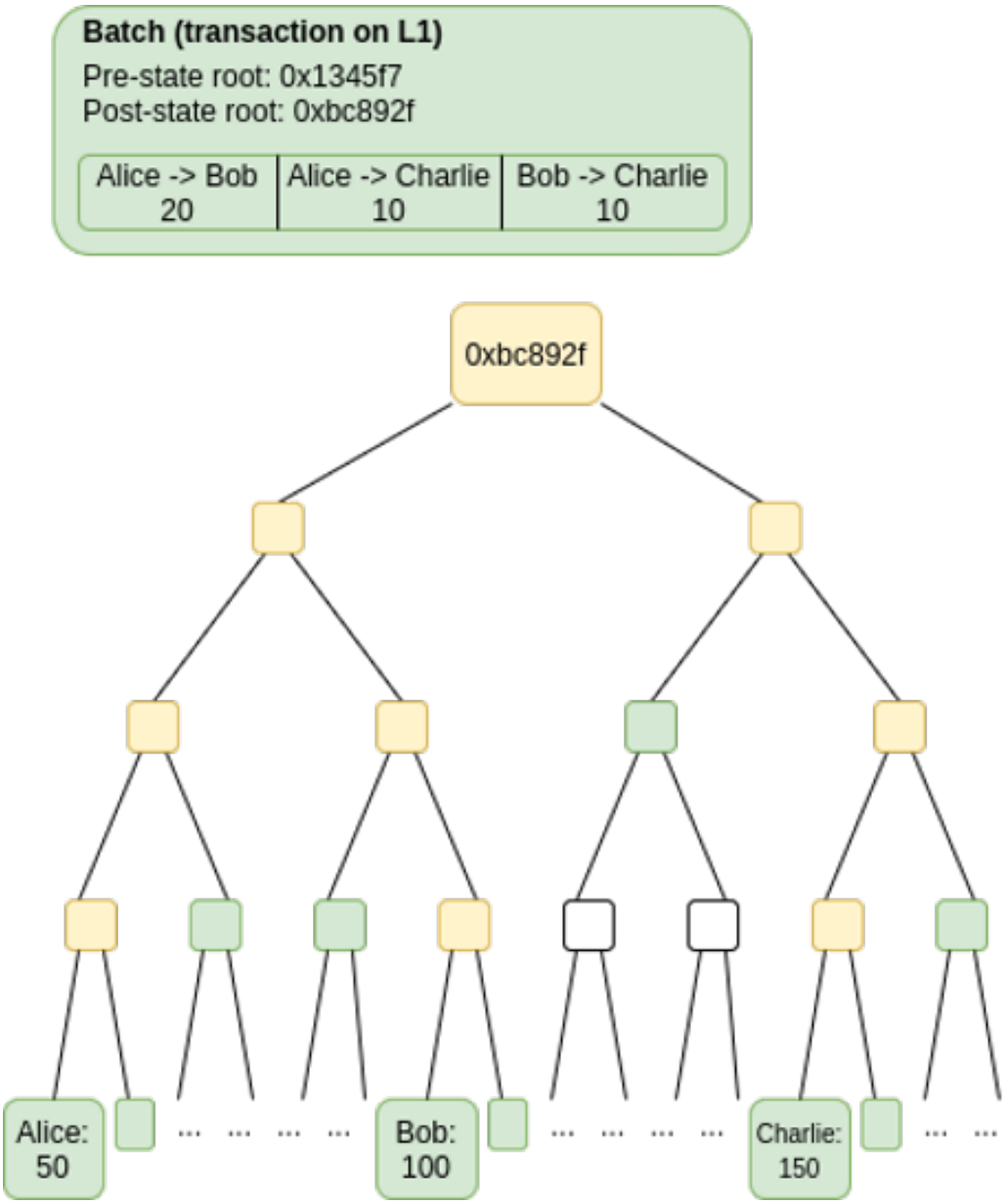


Figure 14.5: Batch transactions on L1: Keeping track of the numbers

with minimal computation.
Arbitrum's fraud proofs are designed to be computationally light and easily verifiable. They do not involve re-execution but rather rely on cryptographic proofs and aggregated state updates. This significantly reduces gas costs and overhead associated with dispute resolution.

In summary, Optimism and Arbitrum address the complexity and overhead of fraud proofs by leveraging innovative execution models and cryptographic techniques. These approaches aim to strike a balance between security and efficiency, enabling Layer 2 solutions to provide scalability benefits without sacrificing the security of on-chain transactions.

14.3.2 Data Availability

Data availability is a crucial aspect of Layer 2 solutions, as ensuring that all necessary transaction data is accessible is essential for the proper functioning of these systems. Here are some methods to address data availability:

1. **All Compressed Transaction Data is On-Chain:** To ensure data availability, all compressed transaction data is stored on the main Ethereum chain. This approach guarantees that the necessary data for transaction execution is accessible to all participants. Storing the minimum required data on-chain allows Layer 2 solutions to operate effectively and maintain security.

2. **Clever Compression Tricks:** Efficient data compression techniques can further optimize the storage and accessibility of transaction data on-chain.

   (a) **Use a Fixed Fee Level in Each Batch:** By employing a fixed fee level for transactions within each batch, the need to include fee-related data for every transaction can be minimized. This helps reduce the overall data size and ensures data availability without overburdening the system with redundant fee information.

   (b) **Replace the 20-byte Address with an Index:** Substituting the standard 20-byte Ethereum addresses with shorter indices can significantly reduce the data size of transactions. This optimization relies on maintaining an index-to-address mapping off-chain, which helps achieve efficient data storage while still allowing easy address resolution.

   (c) **Use BLS Aggregate Signatures:** Utilizing BLS (Boneh-Lynn-Shacham) aggregate signatures allows multiple signatures to be combined into a single compact signature. This reduces the overall data size required to represent the signatures of multiple participants in a transaction batch, improving data availability and on-chain efficiency.

   (d) **Write Transactions to Ethereum as Calldata:** Storing transaction data in Ethereum calldata, which is a more cost-effective data storage location compared to storage slots, can further optimize data availability. Calldata storage reduces gas costs and ensures efficient on-chain transaction data representation.

In summary, addressing data availability in Layer 2 solutions involves storing compressed transaction data on-chain and implementing clever compression techniques to optimize the use of data storage. These approaches collectively ensure that the necessary transaction data is accessible, while simultaneously minimizing the associated storage costs and data overhead.

14.3.3 Pros and Cons

In this part we provide a simple table to show pros and cons of this approach, see table 1.

14.4 ZK Rollup

Zero-Knowledge Rollups (ZK Rollups) are a Layer 2 scaling solution for blockchains that aims to improve the efficiency and scalability of transactions while maintaining a high level of security. ZK Rollups leverage the power of zero-knowledge proofs to achieve significant throughput improvements without compromising the security guarantees of the underlying blockchain.

Table 14.1: Pros and cons of Optimistic Rollup

pros	cons
Offers massive improvements in scalability without sacrificing security.	Security model relies on at least one honest node executing rollup transactions and submitting fraud proofs.
Permissionless (anyone can force the chain to advance by executing transactions and posting assertions)	Users must wait for the one-week challenge period to expire before withdrawing funds back to Ethereum
Compatibility with EVM and Solidity allows developers to port Ethereum-native smart contracts to rollups	Rollups must post all transaction data on-chain, which can increase costs.

14.4.1 Validity Proof

In ZK Rollups, a validity proof is generated using ZK-SNARKs to ensure that the state transition from the pre-state to the post-state has been executed correctly based on the provided transactions.
Components of the Validity Proof:

- **C - State Transition Program:** The program C represents the state transition function or program. It describes how the blockchain state should change based on the given transactions. This program ensures that the pre-state is correctly updated to the post-state according to the specified rules.
- **x - Public Input:** The public input x consists of the pre-state and the post-state. It provides information about the initial state of the blockchain and the desired final state after processing the transactions.
- **w - Private Input:** The private input w comprises all the individual transactions that are being processed in the state transition. These transactions are kept private, and their details are used to generate the proof of the correctness of the state transition.

Generating the Validity Proof:
The ZK rollup coordinator, which is responsible for managing the Layer 2 solution, generates a SNARK proof using ZK-SNARK technology. This proof π demonstrates that the coordinator knows the private transactions (input w) that lead to the correct update of the post-state from the pre-state according to the state transition program C.
By generating this validity proof, the ZK rollup coordinator provides cryptographic evidence that the state transition has been executed correctly, without revealing the specific details of the transactions. This proof can then be submitted to the main blockchain (Layer 1) for verification.
The use of ZK-SNARKs in ZK Rollups enables the creation of succinct and efficient proofs that demonstrate the integrity and validity of the state transition. It allows for substantial scalability improvements by reducing the computational load on the main blockchain while maintaining the security and trustworthiness of the entire system.

14.4.2 Data availability

While validity proofs in ZK Rollups ensure the correctness of state transitions and verify the knowledge of transactions, they do not inherently guarantee the availability of data related to those transactions on the blockchain. This is where the concept of "Transaction Summary" (TX summary) comes into play, see Figure 6.
What is Transaction Summary (TX Summary)?
The Transaction Summary (TX summary) is a mechanism used to ensure the availability of transaction data on the blockchain. It provides a way to prove that the necessary data for processing transactions, such as input and output values, exists and can be reconstructed from the blockchain data. The TX summary acts as a complementary layer to the validity proofs, ensuring that all transaction details are accessible and can be verified by any party.
Why is TX Summary Needed?

1. **Data Availability:** The main purpose of TX summary is to guarantee the availability of transaction data. While validity proofs ensure the correctness of state transitions, they don't prove that the data used in

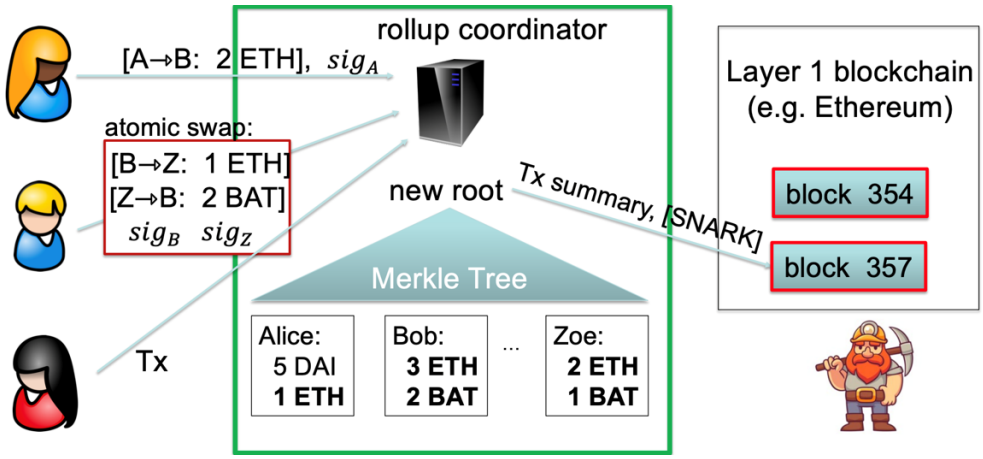


Figure 14.6: Data Availability: Ensuring the validity and authenticity of transactions with Merkle trees and snarks

those transactions is available to all participants. Without data availability, participants wouldn't be able to reconstruct and validate the state transitions themselves.

2. **Resilience:** By including transaction data in a TX summary, the blockchain network becomes more resilient to data loss or unavailability. In cases where some transaction data is missing from the blockchain, participants can still reconstruct and validate the transactions using the TX summary.

3. **Auditing and Verification:** The TX summary allows anyone to audit and verify the details of transactions. It provides a way for users, validators, and other stakeholders to independently verify that the transactions were executed correctly and that the associated data is present and retrievable.

4. **Decentralization:** TX summary contributes to the decentralization of data storage. Instead of relying on a single party or entity to store all transaction data, participants collectively ensure that the data is available and can be used for verification.

In essence, the TX summary complements the validity proofs by ensuring that all necessary data for transaction execution and validation is available and can be reconstructed. It addresses potential issues related to data unavailability, enhances network resilience, and enables independent verification by participants.
It's important to note that different Layer 2 scaling solutions may implement TX summaries in different ways, and the specific technical details may vary. However, the underlying goal remains consistent: to ensure the availability of transaction data for proper verification and validation.

14.4.3 zkSync vs. zkPorter

These approaches focus on different methods of ensuring data availability and managing transaction summaries in a ZK Rollup setup. Here's a bit more detail on each approach:
zkSync:

- **Transaction Summaries on Ethereum:** In the zkSync approach, transaction summaries are stored directly on the Ethereum blockchain. A crucial feature of zkSync is that Ethereum only accepts a transaction batch if it includes the summary of all transactions within that batch.

- **Reconstruction of L2 State:** Other coordinators can use the transaction summaries on the Ethereum blockchain to reconstruct the Layer 2 (L2) state. This ensures that all necessary data for transaction validation is available and can be verified independently.

- **Higher Ethereum Transaction Fees:** One downside of this approach is that storing transaction summaries directly on Ethereum can result in higher transaction fees, especially during periods of network

congestion. This approach is more suitable for high-value assets or use cases where the higher cost is justified by the security and guarantees provided.

zkPorter:

- **New Blockchain for Tx Data:** In zkPorter, the transaction data is stored on a new blockchain that is maintained by a set of staked coordinators. This new blockchain serves as an off-chain storage solution for transaction data.
- **Set of Staked Coordinators:** The new blockchain is secured and maintained by a group of coordinators who have staked tokens as collateral. These coordinators are responsible for ensuring data availability and correct execution of transactions.
- **Cost-Efficient Off-Chain Storage:** zkPorter benefits from cheaper off-chain storage compared to zkSync's on-chain approach. However, it may provide a slightly lower level of guarantee compared to zkSync due to the different security model.
- **Flexibility for Account Storage:** zkPorter allows customers to choose how coordinators store their account data, providing some flexibility in terms of data management.

Both zkSync and zkPorter represent innovative ways to address the data availability challenge in ZK Rollups. They offer different trade-offs in terms of transaction fees, security, and guarantees, allowing projects to choose the approach that best aligns with their specific use cases and requirements.

14.4.4 EVM Compatibility

EVM compatibility is a crucial aspect when considering the adoption of ZK Rollup solutions within the Ethereum ecosystem. Here's a brief overview of the challenges and some of the ongoing projects related to EVM compatibility in the context of ZK Rollups:
Complexity and Proof Generation Challenge: While ZK Rollups can efficiently handle simple computations like token transfers, more complex general-purpose computations that involve smart contracts are harder to prove within ZK-SNARK circuits. The process of creating proofs for complex computations can be resource-intensive and challenging.
zkEVM: Ongoing Projects for EVM-Compatible ZK Rollups:

1. **Polygon (formerly Matic):** Polygon is working on implementing zkEVM, an EVM-compatible ZK Rollup solution. This aims to bring the benefits of ZK Rollups to Ethereum-compatible smart contracts, enabling scalable and low-cost execution of complex computations on the Ethereum network.
2. **Scroll:** Scroll is a Layer 2 scaling solution that focuses on providing EVM compatibility for ZK Rollups. It aims to enable seamless integration of existing Ethereum smart contracts onto Layer 2 while maintaining compatibility with the Ethereum Virtual Machine.
3. **zkSync:** zkSync, as mentioned earlier, is another project that is working on EVM compatibility. It seeks to allow Ethereum smart contracts to run on Layer 2 with the benefits of ZK Rollups, enhancing scalability and reducing transaction costs.
4. **StarkWare:** StarkWare is a well-known name in the field of Layer 2 scalability. While they primarily focus on Stark-based solutions, which use a different zero-knowledge proof technology than ZK-SNARKs, they also contribute to EVM compatibility in Layer 2 solutions.

These projects aim to overcome the challenges of implementing general-purpose EVM computations within ZK Rollup solutions. By providing EVM compatibility, they enable developers to migrate and deploy their existing smart contracts to Layer 2 environments, benefiting from improved scalability and reduced gas fees while maintaining the Ethereum programming model.

14.4.5 zkEVM

zkEVM (Zero-Knowledge Ethereum Virtual Machine) is a technology that brings the functionality of the Ethereum Virtual Machine (EVM) to a Layer 2 scaling solution, specifically a ZK Rollup. It achieves this by recreating existing EVM opcodes within a zero-knowledge proof system, enabling the execution of smart contracts off-chain while ensuring their correctness.
Here's a breakdown of how zkEVM works:

1. **Recreating EVM Opcodes:** zkEVM implements a set of EVM opcodes within the context of zero-knowledge proofs. These opcodes represent the operations that smart contracts can perform on the Ethereum network, such as arithmetic computations, data storage, and control flow.
2. **Computation and State Transition:** Similar to the EVM, zkEVM performs computation on given inputs within a smart contract. This computation results in a transition from one state to another. However, in zkEVM, these state transitions occur off-chain, reducing the computational and gas costs compared to on-chain execution.
3. **Zero-Knowledge Proofs:** The key innovation of zkEVM is the generation of zero-knowledge proofs for every step in the program's execution. These proofs cryptographically verify that the computations performed in zkEVM are correct without revealing any sensitive information about the actual inputs or intermediate states.
4. **Off-Chain Execution:** With zkEVM, smart contracts are executed off-chain within the Layer 2 environment. The zkEVM protocol creates and submits zero-knowledge proofs to the main Ethereum chain (Layer 1), providing cryptographic evidence of the validity of the computations performed off-chain.
5. **Scalability and Efficiency:** By leveraging zero-knowledge proofs, zkEVM significantly reduces the amount of data that needs to be stored and verified on-chain. This results in improved scalability and reduced gas fees compared to directly executing smart contracts on the Ethereum mainnet.

In summary, zkEVM allows developers to migrate their existing Ethereum smart contracts to a Layer 2 environment while maintaining compatibility with the Ethereum programming model. It achieves this by combining the familiar functionality of the EVM with the power of zero-knowledge proofs, enabling efficient and secure off-chain execution of smart contracts with cryptographic proofs of correctness.

14.4.6 Pros and Cons

In this part we provide a simple table to show pros and cons of this approach, see table 2.

Table 14.2: Pros and cons of ZK Rollup

pros	**cons**
Short withdrawal delays	Building EVM-compatible ZK-rollups is difficult due to complexity of ZK technology
Relies only on trustless cryptographic mechanisms for security	High cost on computing and verifying validity proofs
Only TX summaries on chain	Some proving systems (e.g., ZK-SNARK) require a trusted setup

14.5 Final Comparison

In this part we bring a table to compare ZK and Optimistic Rollups approaches, see table 3.

Table 14.3: ZK Rollups vs. Optimistic Rollups

Property	**Optimistic Rollups**	**ZK Rollups**
Fixed gas cost per batch	40,000	500,000
Withdrawal period	1 week	Very fast
Complexity of technology	Low	High
Generalizability	Easier	Harder
Per-transaction on-chain gas costs	Higher	Lower
Off-chain computation costs	Lower	Higher

Chapter 15

Blockchains with Finality

15.1 Review of the longest chain protocol

The longest chain consensus protocol was the main idea of focus and concentration in the previous chapters. This protocol and its modifications we have seen so far, all favor liveness, but are not safe under network partition. The protocol has some advantages and disadvantages in terms of liveness and safety :

- **Liveness** :
  the protocol is live even with minuscule honest hash power. Even a single honest miner with a small hash power can extend the longest chain
- **Safety** :
  The protocol only guarantees safety when the hash power of honest nodes is more than 50% but with two caveats:
  1. **Probabilistic guarantee** : The safety of the confirmed blocks is provided in terms of the probability of deconfirmation ("error").
  2. **Network must be synchronous** : Honest nodes need to update each other regularly on the status of the longest chain, so that they can work together on extending the same chain, avoiding splitting.

In some applications, safety is very important, especially deterministic guarantees (known as **"finality"**) for a confirmed ledger entry, e.g., financial applications. The longest chain protocol (and any of the variations we saw in the previous chapters) cannot provide much relief in this case.

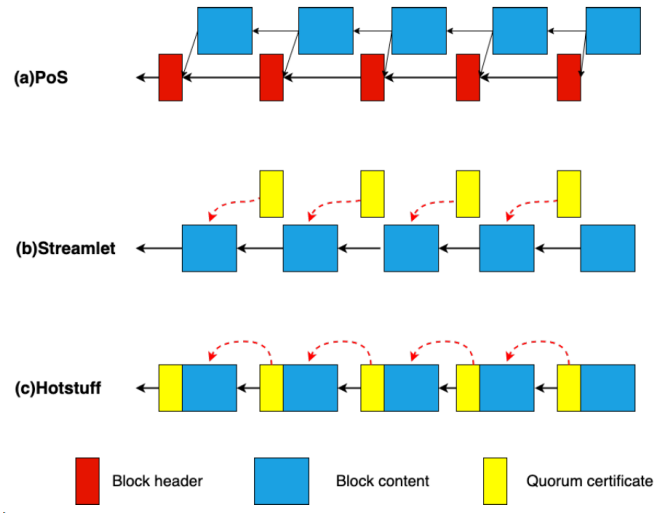


Figure 15.1

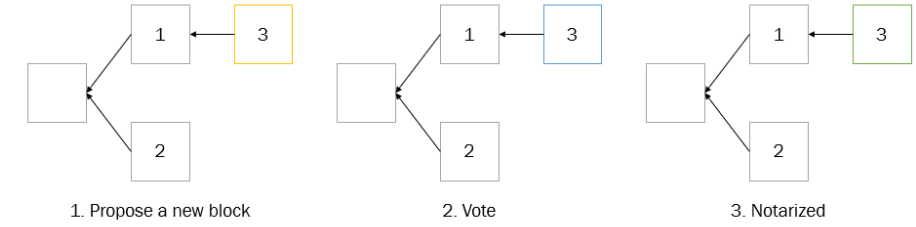


Figure 15.2: BFT Steps (Round-by-Round)

15.2 Byzantine Fault Tolerant (BFT) Protocols

Byzantine Fault Tolerant (BFT) Protocols are a class of blockchain protocols that can achieve deterministic safety even under asynchronous network conditions. This means that they can guarantee that the confirmed ledger entries are final and unchangeable, even if some nodes are malicious or the network is slow or unreliable. Two closely related protocols that belong to this class are **Streamlet and HotStuff**.

The BFT protocol works with a set of nodes (called N) that are fixed and have a fixed identity (public key) that everyone knows. The protocol runs in rounds, which are numbered by integers, similar to the PoS longest chain protocols. But unlike the PoS protocols, here only one proposer is picked in each round. The proposer has to be chosen and checked in a distributed way. In a permissioned system, we can pick the proposer in two simple ways:

1. By taking turns, for example, round i's proposer is the node (i mod N)
2. By using a distributed pseudo-random function or a hash function $H : \{0,1\} \rightarrow [N]$

15.3 Streamlet

The protocol works in rounds, and each round has three steps:

1. **propose** : The round's designated proposer proposes a new block extending from the longest notarized chain it has seen (if there are multiple, break ties arbitrarily).
2. **vote** : The first block that a node receives from the leader of the round gets a vote from the node, as long as the block builds on top of (one of) the longest chain(s) that has been notarized by more than one-third of the nodes that the voter knows. A vote is a signature on the block.
3. **notarize** : A block becomes notarized when more than two-thirds of the nodes vote for it. A chain is notarized when all of its blocks are notarized.

The protocol repeats these steps for each round until a final and consistent ledger is achieved.

15.3.1 Confirmation rule

We call the set of distinct votes on a notarized block a **quorum certificate (QC)**. The quorum size has to be more than two-thirds of the nodes, so that only one block per round can be notarized if the number of bad nodes is less than one-third. This is because two quorums have at least one-third of the nodes in common (see Figure 2) and only bad nodes will vote for more than one block.
The confirmation rule states that a block is confirmed if it has been notarized by more than two-thirds of the nodes, and if it extends a chain that has been notarized by more than one-third of the nodes.

We might think that we can confirm notarized blocks right away, since only one block per round can be notarized. But this is not secure, because there can be two blocks in different rounds that are both notarized at the same level of the blockchain, as shown by the following example.

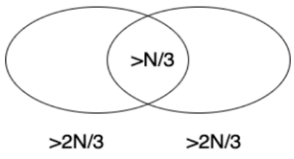


Figure 15.3: The intersection of two quorums must be greater than $\frac{N}{3}$.

Example 15.3.1 (Con
rming a 1-deep notarized block is insecure.)

We cannot confirm a block just because it is notarized. Let f be the number of malicious nodes, and $f < \frac{N}{3}$. In Figure 15.4, a malicious node from round 3 makes a block B_3, but it only sends B_3 to x honest nodes, where $\frac{2N}{3} - f < x < \frac{2N}{3}$. If all f malicious nodes hide their votes on B_3, then no honest node will see B_3 as notarized. Suppose the leader of round 4 is good, then it will make a block B_4 at the same level as B_3. All good nodes vote for B_4, and it becomes notarized. After that, all f bad nodes show their votes on B_3 and make B_3 notarized. Since B_3 and B_4 are different (not on a chain), confirming either one of them will break safety. So notarization is not enough for confirmation.

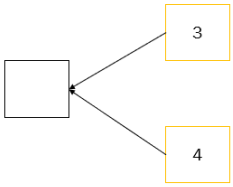


Figure 15.4: Con
rming a 1-deep notarized block is insecure

Confirming a block that is k-deep in the longest notarized chain is not safe either. When the network is not fast or reliable, the adversary can create two chains that are equally long and have any length, by using the same trick as before at every level. We assume that the adversary can make any message take as long as they want when the network is slow or unreliable. In Figure **??**, at each level, the adversary makes sure that only x honest nodes vote for the block (with an odd round number) in the upper chain, where $\frac{2N}{3} - f < x < \frac{2N}{3}$. And after the different block (with an even round number) in the lower chain becomes notarized, the adversary shows f hidden votes to make the block in the upper chain notarized. Because of this balance attack, confirming the block that is k-deep in the longest notarized chain is still not safe in Streamlet.

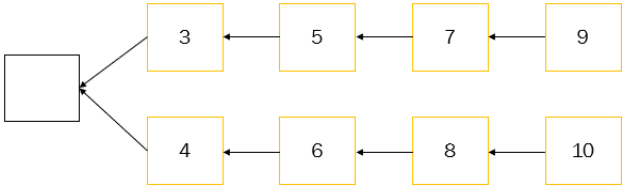


Figure 15.5: Con
rming a k-deep notarized block is also insecure.

Confirming a 3-deep notarized block with consecutive round numbers is secure. The correct rule for confirming blocks in Streamlet is this: if there are three blocks next to each other in a notarized chain, and they have round numbers that go up by one, then we can confirm the chain up to the middle block of the three.

This rule is safe! Let's say one good node sees three notarized blocks B_5, B_6, B_7 from rounds 5, 6, 7 in a chain (see Figure 15.6), to show that it is safe we will argue that no other block can be notarized at the same level as B_6. To get a contradiction, let's suppose there is a notarized block B from round X that is different from B_6. Because only one block per round can be notarized, we know X is either bigger than 7 or smaller than 5.

- **Case 1**: $X < 5$. Since block B is notarized, it means that more than one-third of the honest nodes, which we call the set S, voted for block B and also saw block B_3 notarized when they voted (that is, during round X which is smaller than 5). Now the good nodes in S will not vote for block B_5 in round 5, because it does not build on top of the longest notarized chain they saw, which is block B_3 or longer. Since $f < \frac{N}{3}$, this means that block B_5 can never be notarized by any good node, which is a contradiction.
- **Case 2**: $X > 7$. Since block B_7 is notarized, more than one-third of the good nodes (which we call the set S) must have seen a notarized block B_6 before they voted for block B_7 (that is, by the end of round 7). So in round X which is bigger than 7, the set S of nodes must have seen block B_6 notarized and will not vote for block B, because block B does not build on top of the longest notarized chain they saw (which is block B6 or longer). Since $f < \frac{N}{3}$, this means that block B can never be notarized by any good node, which is a contradiction.

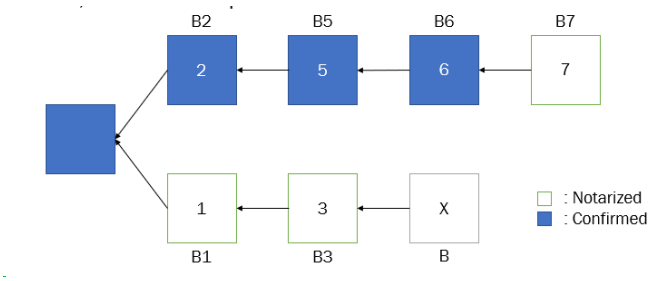


Figure 15.6: Streamlet con rmation example.

15.3.2 Streamlet performance

Communication complexity and **Confirmation latency** are two criteria we will discuss to measure streamlet performance.

Communication complexity of Streamlet

Streamlet makes nodes repeat every message they get (blocks or transactions) to everyone else. Let's say a block has B bits and a vote has V bits. In each round, there are N senders and N receivers, and on each link one block and N votes are sent (because of repeating). The total communication is $N^2(B + NV) = N^2B + N^3V$ bits per block, even when the leader is good. If N is big (that is, we want the protocol to work well with many nodes), N^3V is the biggest part of the communication cost. Also, Streamlet needs repeating for its security (so it has to pay N^3 communication cost);

Example 15.3.2

Let's say we have a blockchain system with 7 nodes: a, b, c, d, e, x, and y. Nodes x and y are malicious. By round r, all honest nodes see the same thing. Then, x is picked to be the leader of round r. In the first half of round r, it only sends its block B to nodes a, b, c and y. In the second half of the round, a, b and c follow the rules and vote for B, sending their signature to everyone. The other malicious node, y, only sends its vote to a. At this point the round is over and no more votes on B are allowed. We can see that at this point a has 5 votes for B (from a, b, c, x, and y), which is more than two-thirds of the nodes. But any other good node only has 4 votes for B. This means that only a sees B as notarized. If there are no vote repeats, in the next rounds a does not vote for any block that does not build on B. If a and y stop working, the system cannot make progress. When a is the leader, it tries to extend B but no one else votes for it, because they don't know it is notarized. So the chain stops growing. See Figure 15.7

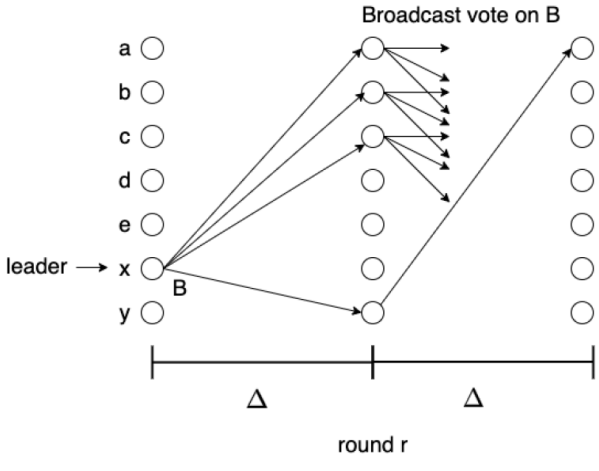


Figure 15.7: Communication Complexity

A block needs at least $N-1$ messages to reach all nodes, so the lowest communication cost per block is $O(N)$. A blockchain protocol is linear if its communication complexity is proportional to the number of nodes. Streamlet is far from being linear because it uses broadcasting and echoing.

15.3.3 Latency of Streamlet

To guarantee liveness, Streamlet makes an assumption that rounds operate in lock-step with 2Δ duration. When there is no forking or adversary, Streamlet can confirm transactions in two rounds (or 4Δ in time). Even if there are $f < \frac{N}{3}$ malicious nodes, Streamlet can still achieve liveness as long as there are 5 honest proposers in a row. This happens on average every 7.6 rounds (or 15Δ in time), assuming that $\frac{2}{3}$ of the nodes are honest.

Recall that Bitcoin latency is $O(\ln(\frac{1}{\epsilon})\frac{1}{\lambda\Delta})\Delta$, where ϵ is the confirmation error probability and $\lambda\Delta \ll 1$ for security. So compared to Bitcoin, Streamlet achieves much better latency: small constant and finality (i.e., deterministic confirmation).

However, BFT protocols can actually achieve even better latency by having non lock-step rounds. Streamlet sacrifices a key feature of asynchronous consensus protocols, which is responsivity. This means that the protocol can adapt to the actual network speed and not wait for the worst-case network delay.

Streamlet does not store the votes on the blockchain itself. This means that Streamlet can create a total-order on transactions, but it does not solve the problem of how an external client (who is not part of the permissioned committee) can check the validity of the ledger and confirm a transaction. This is an important issue that Streamlet does not address.

15.4 HotStuff

HotStuff is a state-of-the-art BFT consensus protocol that can achieve high performance and scalability in distributed systems. It is similar to Streamlet, but it was proposed earlier and has some advantages over it :

- HotStuff is **linear**, which means that it only requires a constant number of communication steps per decision.
- It is also **responsive**, which means that it can adapt to the actual network speed and not wait for the worst-case network delay.

HotStuff assumes that there are n nodes in the system, of which f can be Byzantine (faulty or malicious). To reach consensus, HotStuff requires a quorum certificate (QC), which is a set of 2f+1 votes on one block from different nodes. QCs are stored on the blockchain, which is a linear chain of blocks.
HotStuff uses quorum certificates (QC) to link each block to its parent block, creating a linear chain of blocks. A

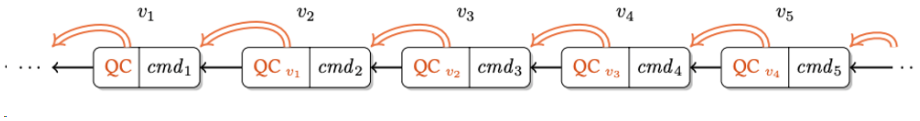


Figure 15.8: each node proposes a block with a command (cmd) and a QC, and how other nodes vote on it.

proposer can propose a new block as soon as it receives a new QC, without waiting for the previous block to be decided. This makes HotStuff more efficient and responsive than other protocols that require more communication steps per decision.

15.4.1 From Streamlet to Hotstuff

Every node keeps track of the QC with the highest round number (HighQC) that it knows. Like Streamlet, a proposer proposes a block that extends HighQC, and a node votes for a proposal if it follows the branch of its HighQC. However, a key difference is that a node only sends its vote to the next round's proposer. This reduces the communication cost per block to NB + NV, where N is the number of nodes, B is the size of a block, and V is the size of a vote. Therefore, HotStuff has linear complexity, which means that it scales well with the number of nodes.

HotStuff has the same finalization rule as Streamlet: a block is finalized if it has two QCs on it and its parent, and the rounds of the three blocks are consecutive. This means that the middle block of the three blocks is irreversible (along with all the previous blocks in the chain).

HotStuff does not need round synchronization. A proposer can propose a new block as soon as it gets a new QC, which means that the protocol is driven by events rather than by time, unlike Streamlet. The protocol can advance at the speed of the actual network delays, so HotStuff is responsive. By increasing the time each node spends in each round until a decision is made, liveness is ensured.

15.5 implementation: Pacemaker

Pacemaker, which is a component that elects proposers for each round of the protocol. The pacemaker guarantees two properties:

1. eventually all nodes will be in the same round for a certain period of time
2. there will be a unique correct proposer for that round.

A naive way to implement the pacemaker is to double the round size until a decision is made, and to rotate the proposer among the nodes in a round-robin fashion. However, this approach has some drawbacks, such as high latency and vulnerability to network delays.

Conclusion

Finality means that once a transaction is confirmed, it cannot be reversed or changed. Streamlet and HotStuff are two examples of such protocols, which can achieve high performance and scalability in distributed systems. They are permissioned protocols, which means that they have a fixed number of participants with known identities. They also have a strong security guarantee, which is that they can tolerate up to one-third of the nodes being faulty or malicious. However, they trade off liveness for safety, which means that they may not make progress in some situations, such as network delays or adversarial attacks.

Chapter 16

Algorand

16.1 From Permissioned to Permissionless: Committee Selection

Transitioning from a permissioned to a semi-permissionless setup involves addressing the challenge of selecting a committee of variable size (N) to execute a Byzantine Fault Tolerant (BFT) consensus protocol, such as HotStuff, in a distributed and verifiable manner. This committee, chosen based on the randomness of the previous block, plays a critical role in ensuring the safety and liveness of the protocol. If at least $\frac{2}{3}$ of the committee members are honest and online, the BFT protocol remains secure and operational. To account for randomness, a certain degree of slack is allowed in the honest committee member fraction. For instance, if the honest fraction is 0.7, a $\frac{2}{3}$ honest supermajority within the committee can be achieved.
Directly selecting a fixed committee size is complex in a distributed context, but randomness can be introduced to determine the committee size, offering a viable alternative. This can be achieved using hash functions, similarly to the approach described in a previous lecture (Lecture 12). Each player, whether honest or malicious, participates in a lottery where they have a small probability of being elected. Consequently, the committee size (N) becomes random, albeit with a predictable expectation. For instance, if there are $700K$ honest players and $300K$ malicious ones in total, and the expected committee size is $E[N] = 1000$, each player's probability of being elected through the lottery is 0.001. This results in the random variable X following a Binomial($700K$, 0.001) distribution, and similarly, another variable Y follows Binomial($300K$, 0.001), representing malicious committee members.
For a committee with a random size, two conditions are essential:

1. **Ensuring liveness:** $X > \frac{2E[N]}{3}$, guaranteeing that honest players can drive progress within the protocol.
2. **Ensuring safety:** $X + 2Y < \frac{4E[N]}{3}$, ensuring that the committee maintains a $\frac{2}{3}$ honest supermajority.

Upon examining the probability distribution in this context (see figure 2), it becomes evident that the likelihood of a random committee successfully executing a secure BFT protocol approaches certainty as the committee size and the number of participants increase.
However, this form of random committee selection faces security challenges due to potential attacks from adaptive adversaries. These vulnerabilities arise from two scenarios:

- Preknowledge of committee selection, allowing an adaptive adversary to preemptively corrupt committee members.
- Corrupting individual nodes within a committee during the BFT operation, compromising the protocol's integrity.

To mitigate these issues, Algorand introduces innovative solutions:

- A novel committee selection process that remains robust against adaptive adversaries, leveraging secret credentials.
- A novel BFT consensus mechanism that maintains robustness even when individual nodes within the committee are compromised.

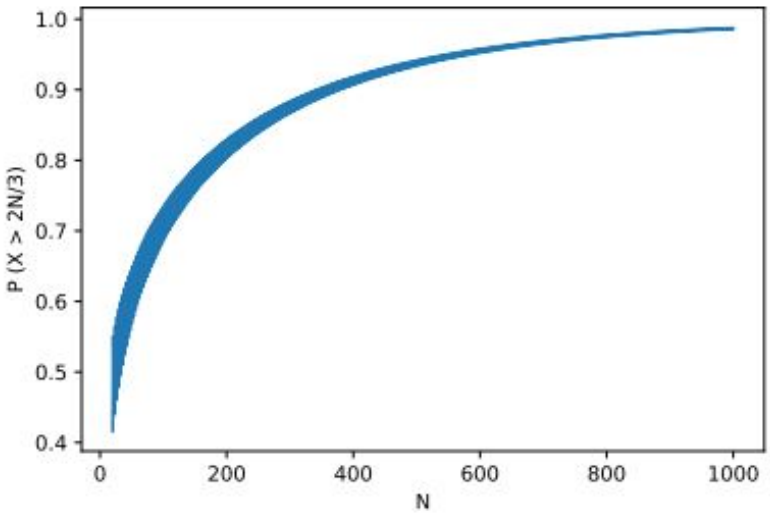


Figure 16.1: Probability of honest supermajority for fixed committee size N

These strategies are elaborated in subsequent sections, addressing the implementation of secure committee selection and BFT consensus protocols, as depicted in figure 3b. Further insights into the execution of these solutions will be discussed in the upcoming sections.

16.2 Player Replaceability and Secret Committee Election

To counter the challenges posed by an adaptive adversary, Algorand introduces two fundamental properties: Player Replaceability and Secret Committee Election. These properties ensure the security and robustness of the Byzantine Fault Tolerant (BFT) protocol. Let's delve into how Algorand achieves these requirements.

Consider a scenario where we are executing a BFT protocol to determine the contents of block B^r associated with round r. To ensure the randomness required for committee selection, a random quantity Q^r is obtained from the previous block B^{r-1} available on the blockchain. This randomness serves as a crucial element in generating secure and unbiased committee selections.

The BFT protocol for block B^r involves multiple distinct steps, each identified by a step counter denoted as 's.' Algorand ensures that different committees are elected for each of these steps. To achieve this, the concept of a "credential" (α) is introduced for player i during any committee selection stage. The credential $\alpha_i^{r,s}$ is derived in secret, utilizing the random quantity Q^r and is a unique signature based on r, s, and Q^r. Specifically, if the result of the hash function H applied to $\alpha_i^{r,s}$ is less than a certain threshold value 'p' ($H(\alpha_i^{r,s}) < p$), then player i is selected as a member of the committee for that step.

The unique secret signature is generated using Verifiable Random Functions (VRFs), which were introduced in Lecture 12. In this context, $\text{VRF}(r, s, Q^r, sk_i) < p$ is used to generate the signature, where sk_i represents the private key for the VRF. The threshold value 'p' is carefully chosen to ensure an appropriate expected committee size.

As the BFT protocol progresses and it becomes time for player i to participate in step 's,' player i includes their credential $\alpha_i^{r,s}$ with their message. Other players can then verify this inclusion, even if player i is maliciously corrupted. Importantly, the message cannot be prevented from reaching other honest players. Furthermore, any influence the adversary might gain from corrupting a player is limited; they have no more control over the protocol's progression than they would have by corrupting a randomly chosen player.

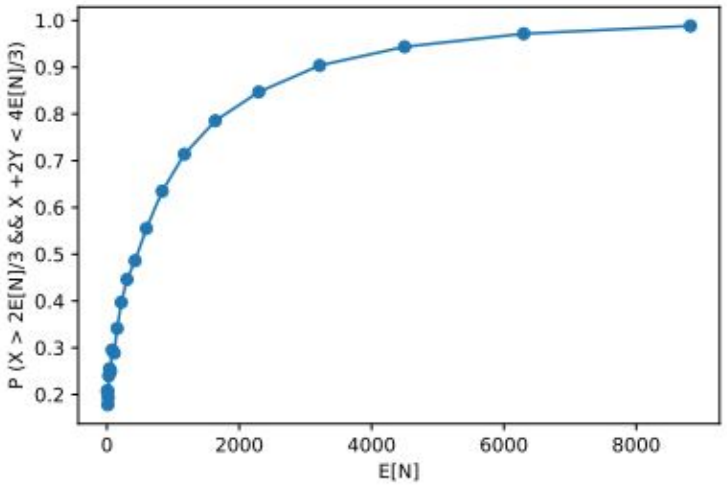
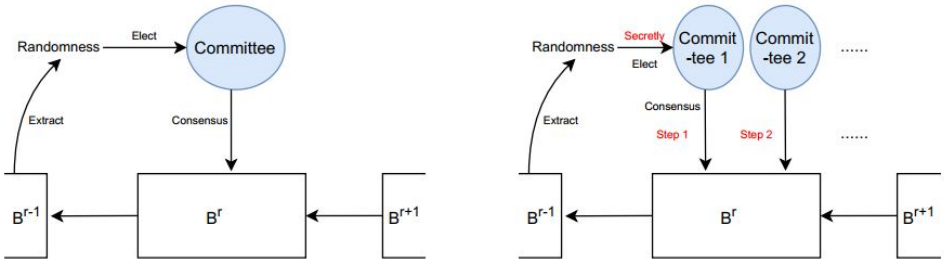


Figure 16.2: Probability of liveness and honest supermajority for a random committee size N. Total players $n = 1000000$ and honest fraction 0.7.



(a) Simple committee

(b) Algorand

Figure 16.3: Simple committee selection vs Algorand committee selection

This process of secret committee election, utilizing credentials and VRFs to generate verifiable signatures, effectively achieves Player Replaceability. Each substep of the BFT protocol has its own randomly and independently selected committee, making the protocol resistant to preknowledge-based attacks. This ensures that an adaptive adversary cannot exploit committee predictability or corrupt members preemptively. The innovative combination of Player Replaceability and Secret Committee Election fortifies Algorand's BFT protocol against adaptive adversaries, preserving the protocol's security and integrity.

16.2.1 Ephemeral keys

To counter the potential threat of an adaptive adversary who could corrupt committee members after learning their identities, Algorand employs a mechanism known as ephemeral keys. These keys are one-time-use public/secret key pairs that provide an added layer of security against the adversary's tactics.
Here's how ephemeral keys work within Algorand's framework:

1. **Generation of Ephemeral Key Pairs:** For each player-round-step triple (i, r, s), player i generates a master key pair. This master key pair is then used to generate ephemeral key pairs for multiple rounds and steps. Importantly, once these ephemeral key pairs are generated and used, the master secret key is destroyed to prevent any future use. The master public key, however, is made public.

2. **Ephemeral Key Usage:** During round r, step s, if player i is elected as a committee member, they use the ephemeral key pair specifically generated for the (i, r, s) triple to sign messages relevant to that committee participation. Once these messages are signed, the ephemeral secret key is destroyed. It's worth noting that ephemeral keys are only used to sign messages, while the long-term public/secret key pairs are reserved for signing credentials α.

3. **Contrast with Key Evolving Scheme (KES):** Ephemeral keys differ from the key evolving scheme (KES) introduced in Lecture 12. In KES, keys evolve from the previous ephemeral key pair, while in the case of ephemeral keys in Algorand, they are generated by a master key pair. The ephemeral key pairs used for a specific (i, r, s) triple are destroyed after use, and a player does not need to retain keys for every round and step unless elected as a committee member.

The use of ephemeral keys provides an additional level of protection against adaptive adversaries. By generating new keys for each committee participation and destroying them afterward, Algorand ensures that even if the adversary identifies committee members based on their messages, they cannot corrupt those members and coerce them into certifying a fake block. This mechanism enhances the security of Algorand's protocol and ensures that the integrity of the consensus process is upheld while also being efficient and well-suited for the permissionless setting.

16.3 Algorand: Single Round BFT Consensus Protocol

We commence with a probabilistic binary Byzantine Fault Tolerant (BFT) consensus protocol that lacks the characteristic of player replaceability. However, it inherently allows for the integration of this feature, as we will elucidate subsequently.
The primary objective of this protocol is to facilitate consensus among a total of $n = 3t + 1$ players, where t represents the threshold for tolerating Byzantine faults. Each individual player, denoted as i, possesses a binary value b_i as their input, which they aim to reconcile. Throughout the course of the protocol, participants continuously update their local binary value b_i based on the information received. Upon conclusion, the protocol strives for a unanimous output among honest players, signifying that they should converge to the same binary value b. If all honest players initially hold the same input value, implying there exists a value b such that $b_i = b$ for every honest player, the protocol's objective is for them to eventually output the same value b, i.e., $b_i = b$ remains valid for all honest players. The protocol operates within synchronized steps, where the delivery of messages is ensured to occur within a designated step. Therefore, the protocol's design aligns with that of a synchronous network, akin to the mechanics of the longest chain protocol. Each step follows the paradigm below:

Start of a step	End of a step
Every player propagates b_i	Every player updates b_i based on the received messages

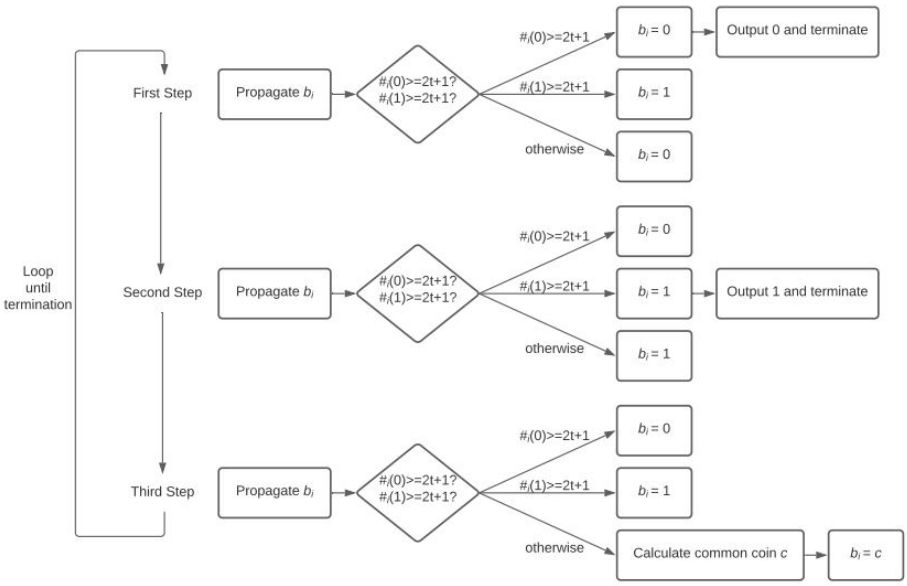


Figure 16.4: Algorand consensus on a binary value

16.4 Consensus On a Binary Value

Algorand's binary Byzantine Fault Tolerant (BFT) consensus protocol operates in a perpetual loop, systematically cycling through three distinct steps. The protocol's structure is depicted in Figure 4, and it employs a counter variable denoted as "s" (initiated at $s = 1$) to signify the number of executed steps. Consequently, the first step corresponds to $s = 1, 4, ...$, the second step to $s = 2, 5, ...$, and the third step to $s = 3, 6, ...$
The first and second steps of the protocol adhere to the previously discussed property (A). Additionally, they are designed to ensure that once consensus has been attained regarding a specific bit, an honest participant can discern this achievement, output the corresponding bit, and conclude their involvement. This property is formally articulated as property (C).
(C) If an honest player "i" outputs a value during the first or second step, then the protocol guarantees that consensus will prevail by the end of that step. Moreover, the third step embodies both property (A) and (B), as previously detailed. By leveraging these properties, the protocol's nature as a Byzantine consensus protocol is established. Specifically, properties (A) and (C) work together to prevent honest players from outputting disparate values, while properties (A) and (B) ensure the eventual convergence and output of a unanimous value, thereby culminating in an agreement.

16.5 Adding Player Replaceability And Secret Committee Election

Incorporating the committee into the protocol involves granting message propagation eligibility exclusively to committee members. However, it's important to note that the fundamental process of updating the local value "b_i" remains a shared responsibility among all players. The overall communication complexity is now quantified as $O(nN)$, attributable to each committee member transmitting messages to every player.
The introduction of player replaceability and secret committee election hinges on the computation of credentials, as elucidated in the previous section. Upon calculating its credential for a specific round "r" and step "s" (denoted as $\alpha_i^{r,s}$), a player discerns its role within the committee and is equipped with the means to substantiate this role. This substantiation is achieved by appending the calculated credential to its messages during the respective step.

In the protocol diagram depicted in Figure 4, a simple adaptation is necessary: the parameter "$2t+1$" is replaced with "$\frac{2E[N]}{3}$." This change is necessitated by the fact that the random committee now boasts an anticipated size of $E[N]$ rather than the original fixed size of $n = 3t + 1$.

16.6 Multivalue Consensus

Extending the Algorand consensus from binary values to multivalued scenarios involves the election of a leader for each round "r," responsible for constructing and distributing a valid block denoted as B^r. The process of electing the "potential leader" remains analogous to the committee election, albeit with a significantly smaller threshold. This adjustment leads to the selection of only a few dozen players as potential leaders.

Once potential leaders are identified, they proceed to propagate their respective blocks. Each player evaluates potential leaders and selects the leader whose credential hash is the smallest. Subsequently, they cast their vote for the block hash twice, with the second vote being dispatched only if the first vote receives more than two-thirds of first-vote responses.

After the block proposing and two-round voting stages, each player reaches one of two conditions: either (a) they receive a valid block B^r from the leader along with sufficient votes for its hash, or (b) no valid block garners enough votes for its hash.

In the case of condition (a), player i initiates the binary protocol with an initial value $b_i = 0$. Conversely, in condition (b), the initial value is set to $b_i = 1$." During the binary protocol, players append the block hash to their messages, ensuring alignment on the held block in condition (a). The two-round voting process further guarantees consensus among honest players in condition (a), ensuring agreement on the same block B^r.

The binary protocol's outcome provides essential insights: a value of 0 signifies the finalization of block B^r, which is universally acknowledged. Conversely, a value of 1 signifies the finalization of an empty block B^r_ϵ.

It's worth noting that the block proposing and two-round voting steps, akin to previous stages, remain both committee-based and player-replaceable. This overarching approach imbues the entire Algorand protocol with committee-based dynamics and the flexibility of player replaceability.

16.7 Conclusion

In the Algorand consensus protocol, players collectively agree on the contents of each block B_r for round r. The Algorand Blockchain is constructed by executing the consensus protocol round by round, sequentially generating blocks B^1, B^2, and so on, with each block containing a hash pointer to its parent block B_r. Each round involves communication complexity of $O(nN)$ and concludes within a constant expected number of steps, as detailed in previous sections.

Algorand can also be adapted into a proof-of-stake (PoS) blockchain by factoring in the stake held by a public key during the committee/leader election process. A higher stake increases the likelihood of being elected. Importantly, Algorand's inherent resistance to forking mitigates issues such as key grinding and nothing-at-stake attacks that plague other PoS mechanisms, as seen in the longest chain version of PoS.

However, Algorand's security model is predicated on a synchronous network setting, where messages are guaranteed to be delivered within a step.

It's important to note that while Algorand offers robust security, there exists a potential weakness related to forensics. In a scenario where the number of Byzantine players exceeds the threshold t, these malicious actors can deviate from the protocol and trigger a safety violation. Unlike some other consensus algorithms, such as HotStuff or Streamlet, Algorand may experience a safety attack that doesn't leave behind cryptographic evidence. For instance, if a significant fraction of Byzantine players abstains from sending votes during the second step, they can cause an honest player majority to switch their local value, leading to a safety breach. Importantly, since these Byzantine players don't send any messages, there's no cryptographic evidence to hold them accountable.

This forensics challenge persists in Algorand regardless of the inclusion of committee elections and player replaceability. The question of whether an efficient BFT protocol with strong forensic support and player replaceability can serve as a core consensus engine within a PoS permissionless blockchain remains an open research question.

References

[1] Silvio Micali. Byzantine agreement, made trivial, 2018.

[2] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science, 777:155183, 2019.*

[3] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics, 2020.

Chapter 17

Longest Chain Protocol Meets BFT

Blockchain protocols must satisfy two basic properties: safety and liveness. They are jointly referred to as security properties. For any given protocol, security properties cannot be guaranteed unconditionally. Naturally, we would like safety and liveness to hold under as large a set of conditions as possible; e.g., safety and liveness guarantees under $\frac{1}{2}$ honest majority would be preferable over these guarantees under $\frac{2}{3}$ honest majority.

One of the main advantages of the (Proof-of-Work) longest-chain protocol is that it operates in a fully permissionless setting. This means that it can handle changes in the number of participants; anyone can join or leave the system at any time. The crucial point here is that it remains secure even if only one participant is active (as long as the majority of the participants are honest at all times). This property is called adaptivity. However, the longest chain protocol also has some drawbacks. For instance, its safety guarantees are not deterministic, but probabilistic. Moreover, the protocol becomes insecure during periods of asynchrony, when the network delays are unpredictable. During such periods, the adversary can easily create many blocks and reverse blocks that are k-deep, thus breaking the safety of the k-deep confirmation rule. It also loses liveness.

We have learned about permissioned (committee-based) BFT protocols in the previous Chapters. These protocols provide deterministic safety (also known as finality). This means that the safety property is always preserved, even when the network is asynchronous for a long time. However, permissioned protocols have a limitation: they lack adaptivity. This means that the protocol will not make progress when the participation level is too low (but safety will never be violated).

Figure 17.1: Two families of Blockchain protocols

We might wonder if we can design a blockchain protocol that has both features: finality and adaptivity. We will explore how finality gadgets can help us create a blockchain system that has two different confirmation rules: one for adaptivity and one for finality. Finality gadgets are fascinating because they cleverly combine a BFT protocol with the longest-chain protocol, to make a protocol that has the best of both worlds. We will also talk about the CAP theorem, which shows us that it is impossible to have a single blockchain protocol that has both features.

There are other ways to compare the longest-chain protocol and the committee-based protocols, besides adaptivity and finality. One important factor is the latency in confirming blocks. The Nakamoto consensus protocol takes a long time to confirm blocks. You have to wait for the blocks to be k-deep before you can trust them. The blocks come at a rate that depends on the maximum network delay Δ. So, the longest-chain protocol has a latency of $O(k\Delta)$.

We have seen how protocols like Prism can improve this to $O(\Delta)$. But protocols like HotStuff can do even better, with a latency of $O(\delta)$, where δ is the actual network delay (which can be much lower than the worst-case scenario). This property is called responsivity; the protocol adapts to the real network delay. Note that not all

committee-based protocols are responsive, for example, Streamlet is not.

We might wonder if we can have a protocol that is responsive in a permissionless setting. This means that the protocol can confirm blocks quickly, even when anyone can join or leave the system. One way to achieve this is by using a protocol design called Hybrid Consensus, which cleverly combines a BFT protocol with the longest chain protocol.

17.1 Hybrid Consensus

The main goal of the Hybrid Consensus approach is to develop a Proof of Work permissionless system with fast confirmation. In Algorand, which uses Proof of Stake, Verifiable Random Functions were used to select a committee, which then ran a fast-confirmation protocol. Inspired by Algorand, the main challenge is to choose a fixed-size committee in a PoW system, and also to do it periodically.

We will start by looking at how to choose one committee of size csize. We will run the Nakamoto consensus protocol until csize + k blocks are mined. Then all parties will agree on the first csize blocks (except with negligible probability). Each block has the miner's public key in it. And there we have it, we have selected a group of csize nodes (or more precisely, csize public keys, which could belong to the same person or entity).

This committee is selected in a fair and decentralized way (because of PoW) and everyone agrees on it (because of the security of the PoW longest chain protocol). Once a committee is selected, it runs a fast and reliable BFT protocol (like HotStuff) to confirm transactions. So, the transactions are actually confirmed by the BFT protocol, not the longest-chain protocol.

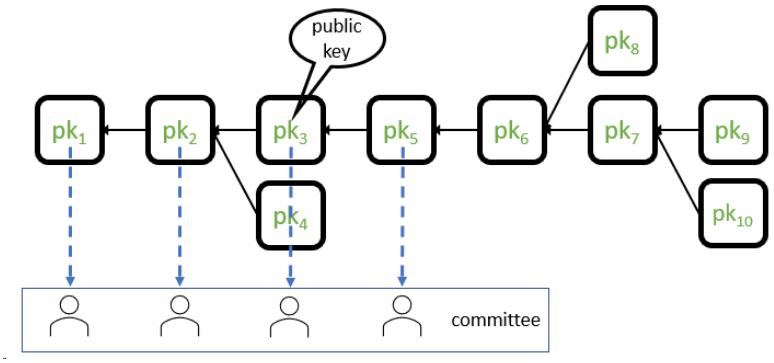


Figure 17.2: Hybrid Consensus

We will now look at the protocol in more detail. First of all, note that honest miners have to keep mining, even if they want to elect only one committee. Otherwise, the dishonest miners can make a longer chain with only dishonest blocks, which will then choose the committee.

Secondly, to run a fast and reliable BFT protocol like HotStuff, we need more than $\frac{2}{3}$ csize honest parties in the committee. This also depends on the ratio of honest and dishonest parties in the PoW protocol. In fact, if we use the simple longest chain protocol, we need a $\frac{3}{4}$ honest majority, because of the low quality of the chain. But, if we use FruitChains, we can do with a $\frac{2}{3}$ honest majority.

We need to change the committee regularly, for two reasons:

1. we cannot expect the committee to stay online all the time for a long time.
2. an adversary can bribe the committee after it is selected (but with some delay)

By changing the committee often, we can prevent attacks by an adaptive adversary. The change is done in a simple way: once a committee is made, it keeps confirming transactions for a fixed amount of time, until the PoW blockchain grows by csize more blocks. When this happens, the old committee passes on the duty of transaction

confirmation to the new committee. Note that everyone in the protocol sees the same committee.

The main advantage Hybrid Consensus brings over the longest-chain protocol is that it is a responsive protocol.
We will now look at some of the drawbacks of Hybrid Consensus.

- First of all, if we want a fast and reliable BFT protocol, the maximum fraction of dishonest parties it can handle is $\frac{1}{3}$, instead of $\frac{1}{2}$. This is unavoidable; any fast protocol can only deal with up to $\frac{1}{3}$ Byzantine adversaries. There are BFT protocols that can deal with up to $\frac{1}{2}$ adversaries, but their latency is $O(\Delta)$, which means that they depend on the maximum network delay.
- Another problem with Hybrid Consensus is that it does not work well under asynchrony, when the network delays are unpredictable. This is because the committee selection mechanism itself loses both safety and liveness. If there is no agreement on who is in the committee, there can be no agreement on the transactions confirmed by them. For the same reasons, the security of the whole system is not deterministic, but probabilistic.
- Another requirement for Hybrid Consensus is that honest committee members must stay active for the whole time that they are part of the committee. If too many of them stop participating, the protocol will stop working.

Thus, in terms of guaranteeing safety and liveness under a wide variety of conditions, Hybrid Consensus actually does worse than the longest-chain protocol it does not guarantee liveness under variable participation.

17.2 The CAP Theorem

Finality and availability are two important properties of blockchain consensus protocols.

- **Finality** means that the transactions that are recorded on the blockchain are irreversible and cannot be changed or reverted by anyone. BFT protocol has this property
- **Availability** means that the blockchain network can continue to operate and process transactions even when some nodes fail or leave. PoW longest chain has this property.

We might wonder if we can have a single ledger that has both adaptivity and finality. Sadly, this is impossible. The reason for this impossibility is a famous result in distributed systems, called the CAP theorem (CAP stands for Consistency, Availability and Partition tolerance).

Roughly speaking, the CAP theorem states that during a network partition, a distributed system must make a choice between availability (liveness) and consistency (safety); it cannot offer both. Moreover, this choice must be encoded in its design itself. Thus, systems can be classified as availability favoring or consistency favoring, based on their design.

The essence of the impossibility result is that it is difficult to distinguish network asynchrony from a reduced number of participants in the blockchain system. Hence, a protocol's behavior must be similar under both these conditions. The CAP theorem also explains why there are two different kinds of protocols in the blockchain

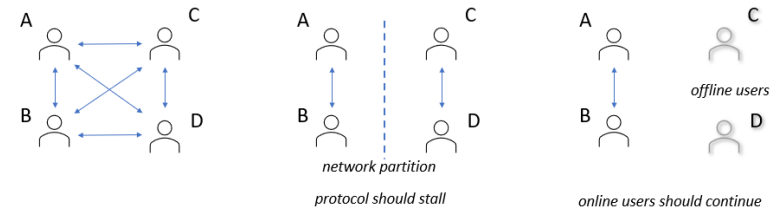


Figure 17.3: A decentralized protocol cannot distinguish between offline users and network partition

world. Protocols that use the longest-chain rule prefer liveness, which makes them adaptive. Protocols that use

a committee of validators (like Hotstuff) prefer safety, which gives them finality.

The CAP theorem lets users choose between adaptivity and finality, depending on their needs. We can use a special gadget that gives us two ways to confirm transactions: one that is adaptive, and one that is final. This way, users can decide locally which one they want to use. For example, if you are buying a coffee, you might not care about finality, and just want a fast confirmation. But if you are buying a Tesla, you might want to be sure that the transaction is final, and not risk it being reversed.

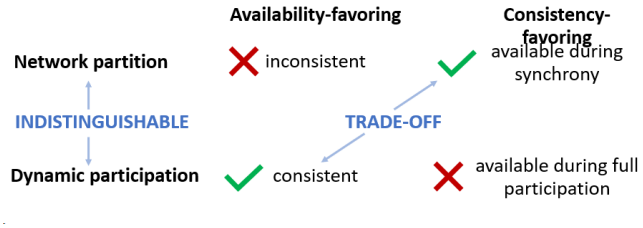


Figure 17.4: CAP Theorem in Blockchains

17.2.1 Proof of CAP Theorem

We can show the CAP theorem with a simple example of a distributed system. Imagine there are two servers, p_1 and p_2. They both have a variable V that has the value x at first. Then, they get separated by a network problem. While they are separated, some client asks server p_1 to change V to y. Now, p_1 has to decide what to do, and it has two options:

- It can say "ok" to the client and change V to y. This means it is available, but p_2 still has V as x. So, when another client asks either p_1 or p_2, they will get different answers, and the system is not consistent.
- It can wait for p_2 to come back online and not say "ok" to the client or change V. This means it is consistent, but not available, because it ignores the client.

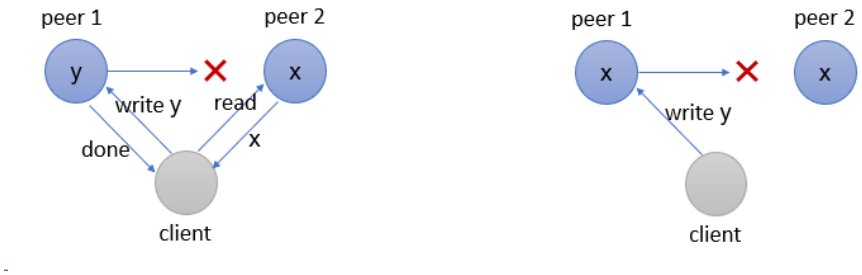


Figure 17.5: A visualization of the trade-offs implicated by the CAP theorem

P_2 also has to make the same trade-off when a client wants to read V. It should try to talk to p_1 to see if V has changed or not. But it can't do that because of the network problem. It can either give an answer after some time (and risk giving the wrong answer) or never give an answer at all. So, if the network is slow and unpredictable (i.e., there is no limit on how long a message takes to reach), the system can't have both consistency (correctness) and availability (responsiveness).

17.3 Finality Gadgets

A finality gadget is a tool that adds finality to a PoW blockchain. Finality means that once a block is confirmed, it can't be reversed. We want the blockchain to have finality and also adaptivity, which means it can adjust to changing conditions. To do this, we can combine two types of protocols: a longest-chain protocol and a BFT protocol. A longest-chain protocol is adaptive, but not final. A BFT protocol is final, but not adaptive. We want to get the best of both worlds, so we need to find a way to mix them together.

17.3.1 A two-layer design

In its simplest form, a finality gadget is a layer-two committee-based BFT protocol, which runs on top of a (layer-one) longest-chain protocol. One can use any adaptive protocol instead of the longest-chain protocol, but for this lecture, we shall focus on the longest-chain protocol. There are two sets of nodes in the system:

- **Miners** : Using PoW and the longest-chain rule, miners create new blocks that contain transactions. The number of miners can vary, but we assume that the system's mining rate is stable.
- **Checkpointers** : Instead of creating any blocks themselves, the checkpointers run a BFT protocol that is based on a committee. They vote on the blocks that the PoW miners have created.

Therefor, to reach consensus on the same set of blocks (transactions), two consensus protocols run at the same time.

Checkpointing

Checkpointing is a new concept that we explain more here. Imagine that the longest chain protocol runs for a while, and the block tree reaches a certain height. Consider a block that has six blocks above it; a user could accept it, hoping that all future blocks would be its descendants. But the acceptance guarantee would be based on chance, as we know. How can we make the acceptance guarantee certain? In other words, how can we checkpoint this block?

The checkpointing committee is responsible for checkpointing blocks, by giving checkpoint certificates for blocks. A checkpoint certificate for a block B is a set of at least $\frac{2n}{3}$ signed messages of the form <finalized, B>; such a block is called checkpointed. These messages are part of almost all committee-based BFT protocols. Any node that gets this certificate can be sure that this block is confirmed for sure. This node could be a miner, a checkpointer, or just a watcher of the system. Of course, the checkpoint blocks, given by the second-level gadget, should be on the longest chain, to keep consistency with the first-level protocol. How can we make this happen?

The checkpointers monitor the blocks that the miners create. At regular intervals, they run a (one-time) BFT protocol, where they input blocks that the miners have made. Honest nodes input blocks that are on the longest chain, but have not been checkpointed yet. Usually, blocks at a certain fixed depth are picked; for example, a depth of 6. This would make sure that if nodes have chains of the same length and have 6 blocks in common at the end, their inputs would be the same. Checkpointers vote on each other's inputs, and finally decide on one specific input; this becomes the checkpointed block.

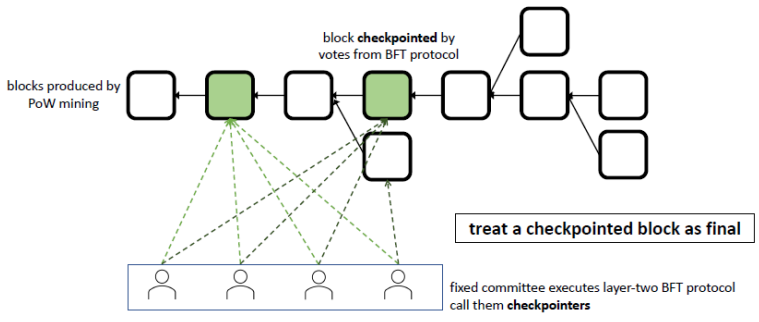


Figure 17.6: Finality Gadget

17.3.2 Two ledgers and their properties.

A system that uses a finality gadget has two different confirmation rule:

- First, the k-deep rule is still a valid confirmation rule in the system, because blocks are mined according to the longest-chain protocol. A node that just watches the blocks mined in the system can easily follow this rule to confirm blocks. Such a node may remain completely oblivious to the checkpointing protocol. This rule provides adaptivity.

- The checkpointers also provide another way of confirming blocks, by giving checkpoint certificates for some blocks at regular intervals. This set of messages proves that block B has been chosen by the checkpointers. The confirmation rule for the finality gadget is to simply confirm the most recent checkpointed block, and all its ancestors. This rule provides finality.

Let's see what adaptivity and finality mean for a confirmation rule. To do that, we need to keep in mind these points:

1. Every confirmation rule creates a ledger: it is the order of all transactions in the blocks confirmed by that rule.

2. Over a long time, the two ledgers are the same: all transactions that are in one ledger are also in the other, and they have the same order.

3. At any given time, one ledger could be slightly ahead of the other (i.e., confirm a few more transactions). Usually, we expect the k-deep rule (for small/moderate k) to confirm blocks faster than the checkpoint-based ledger. So, the ledger made by the k-deep rule will be a bit ahead of the checkpoint-based ledger.

Under optimal conditions (i.e., static and synchronous), both ledgers grow at the same rate. Under variable participation, the finality-preserving ledger stalls, while the adaptive ledger keeps growing. As soon as static participation returns, the finality-preserving ledger catches up.

Two confirmation rules

At first, it may seem strange to have more than one confirmation rule, but it makes sense if we think about it. In fact, the longest-chain protocol already has many confirmation rules! To see this better, let us separate the block production rule and the confirmation rule of the protocol. The block production rule is to make a new block at the end of the longest chain you have seen so far; this is the rule that gives the longest-chain protocol its name. The confirmation rule is to confirm a block that is k blocks deep, and all the blocks before it. All players follow the same block production rule. But, each user can pick their own confirmation rule, i.e., they each pick their own k value, depending on how much security they want and how much delay they can accept.

Validity conditions

We need to be careful about how we pick the inputs for the BFT protocol and how we decide which inputs are valid. Some validity rules are needed, or else bad checkpointers could suggest random blocks for checkpointing, and these might get checkpointed. A validity rule would help honest nodes ignore invalid blocks. Choosing the right validity rules is a hard design problem. For example, we want checkpoints to be for fairly new blocks. Checkpointing an old block is not very useful, because it will already be confirmed with very high probability using the k-deep rule. But, we should not confirm a block very close to the end, because the longest-chain may change from there. Checkpointing a d-deep block, for some moderate value of d, is a possible compromise.

Permissioned or Permissionless?

The checkpointers are a set of nodes that need permission to join in the current design. Can we make the checkpointing process open to anyone? In theory, they could be changed regularly from a large pool of players using a PoS method, like Algorand does. But, some degree of a permissioned system is unavoidable. A pure PoW system cannot give the finality properties we want (the best we could do was what Hybrid Consensus did, but that does not meet our needs). For simplicity, in this chapter, we just assume that the checkpointers stay the same for the whole protocol.

17.3.3 Examining Adaptivity and Finality

We wanted to use finality gadgets to combine a longest-chain protocol and a committee-based BFT protocol to get a system that has both adaptivity and finality. The system design gives us two ways of confirming blocks, and makes us think that the k-deep rule would give adaptivity, and the checkpoint-based rule would give finality. Let's see if we really achieve this.

The k-deep rule is adaptive in our two-layer design, because blocks are mined the same way as they are in the normal longest-chain protocol. The second-layer checkpointing protocol does not affect the first-layer protocol. How does the checkpointing protocol work under variable participation? If not enough checkpointers are active, the protocol simply stops. When all honest checkpointers are online again, they continue checkpointing blocks. So, under variable participation, the checkpointing protocol turns off and on for random periods, depending on how many checkpointers are participating. This only affects the liveness of the protocol; the safety is always preserved.

We want to see if the finality gadget can protect the protocol when there is asynchrony. That means, even after a period of asynchrony, all blocks until the last checkpointed block should stay confirmed. We do not expect that any new blocks will be checkpointed in this period. When synchrony comes back, the protocol should resume checkpointing new blocks, and thus regain liveness. Is this requirement met by the design we have described? It does not work. Imagine a long period of asynchrony. The adversary can make a new chain that splits from before the last checkpointed block, and becomes the longest chain. When synchrony comes back, all miners will see the adversarial chain as the longest chain and will mine on that. There will be no new blocks after the last checkpointed block! At this point, the checkpointers have to either stop completely or (deliberately) break safety by changing chains. Both of these are bad.

17.3.4 checkpointed longest chain rule

This problem happens because we have a two-layer solution. This means that the checkpointers do not affect the miners' behavior. For a finality gadget to work well, miners must follow the checkpointed blocks. In particular, miners should mine after the latest checkpointed block. Of course, we also want some longest-chain-like properties.

A good idea is that miners follow the checkpointed longest chain rule: add to the longest chain after the latest checkpoint block. Since the latest checkpoint block is also the highest checkpoint block, this rule could also be said as: mine on the longest block that has all the checkpoint blocks. Under synchrony, all checkpoints would be on the longest chain by design. Following the checkpointed longest chain rule would be the same as following the longest chain rule. So, the new protocol would keep the security guarantees of the old protocol.

On the other hand, the checkpoints protect the system during a long period of asynchrony. If the adversary mines blocks going back before the previous checkpoint, all honest nodes would simply ignore such blocks as invalid. For all practical purposes, every new checkpoint acts like a new starting point.

We need to be careful about how we design the checkpointed longest-chain protocol, because the checkpointing protocol influences the miners. One vulnerability is how the checkpointing protocol behaves during variable participation. It is possible that the checkpointing protocol stops for a very long time, because of low participation. When participation comes back, we want the protocol to start checkpointing quickly, and close to the end of the longest chain. At all times, the protocol must make sure that the checkpoints are always on the longest chain. To do this, we need to pay special attention to the validity rules we mentioned before.

Chapter 18

Data Privacy via Zero Knowledge Cryptography

18.1 Introduction

In Bitcoin, the state or ledger of the system is maintained using the UTXO format, where each transaction has a list of inputs and outputs. The outputs indicate the public keys of the recipients, and the inputs refer to the outputs of previous transactions. Each UTXO has a history of transactions that can be traced back to its origin. The Bitcoin network has a feature of pseudonymity, which means that the only thing that identifies an account is

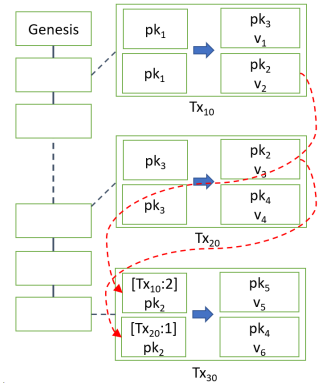


Figure 18.1: UTXO Transactions

the public key. A user can create many public keys to hide their identity. However, the transactions reveal some information about the connections between public keys.
We can use the public keys of the inputs and outputs of a transaction to create a graph that shows the links between transactions. (See Figure 18.2)

18.1.1 Trusted third-party mixer

A Trusted Third-party Mixer (also known as a ”laundry service”) is a service that helps users to protect their privacy on the blockchain. A mixer exchanges the coins (the public keys) of different users, so that the transaction graph cannot trace the public keys.

For example, if a user wants to make a new transaction with some outputs from previous transactions, the UTXO system needs the user to specify which outputs they will use. The mixer can exchange these outputs with other users' outputs, so that the public keys are different.

A mixer is a centralized third-party, which means that it is controlled by a single entity. The mixer can trace or steal the coins of the users, so the users have to trust the mixer.

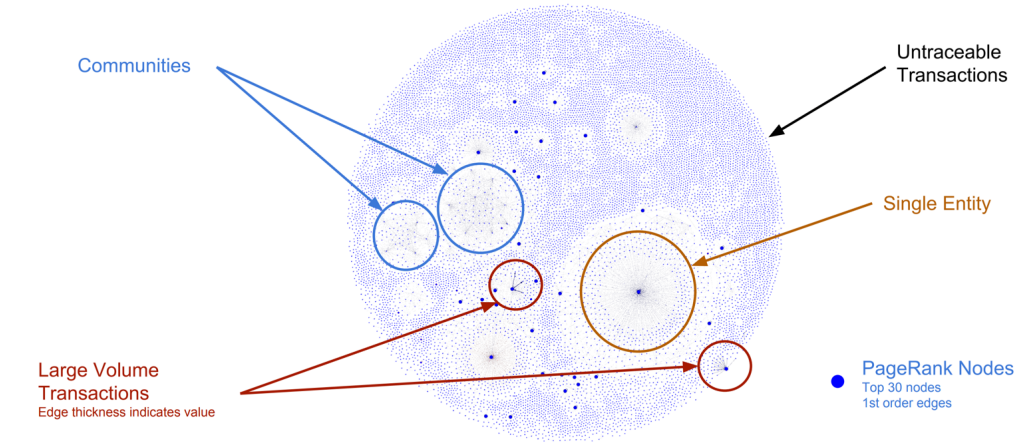


Figure 18.2: Typical transaction graph for a day

Example 18.1.1 (Coinbase)

An example of a mixer is Coinbase, which is a platform that allows users to buy, sell, and store cryptocurrency. Coinbase offers different public keys for each transaction, so that the users can hide their identity.

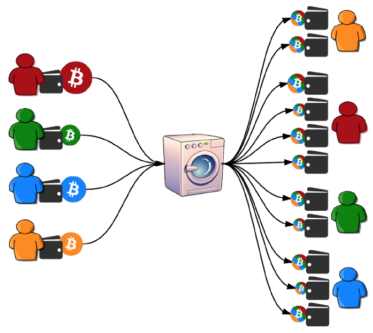


Figure 18.3: Trusted third-party mixer

18.2 Zero-knowledge proof

To achieve a decentralized privacy service that is compatible with the UTXO format, we can eliminate the need for a trusted party in the laundry system. We can do this by using a special UTXO transaction that has an input and an output. Instead of linking the input to a specific previous output, we attach a proof to the input. The proof is valid if it can show that the sender owns the input coin and has not spent it. However, the proof does not reveal which output the input coin comes from.
The property of not revealing connections between inputs and outputs can be achieved by zero-knowledge proofs

18.2.1 Zerocoin

Zerocoin is a protocol that enhances the privacy of Bitcoin transactions. Zerocoin allows users to convert their bitcoins into zerocoins, which are anonymous tokens that can be redeemed for bitcoins later. Zerocoins are stored in a separate ledger on the blockchain, and their origin and destination are hidden by using **zero-knowledge proofs**.

Zero-knowledge proofs are cryptographic techniques that allow users to prove that they own some zerocoins

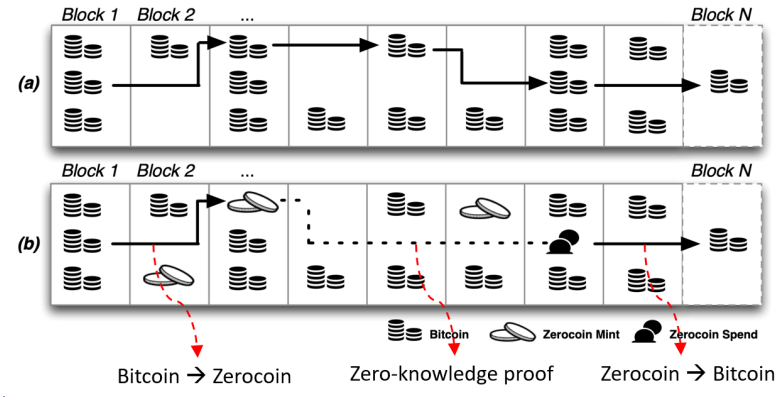


Figure 18.4: Zerocoin

without revealing which ones.Zerocoin is a decentralized laundry system, which means that it does not rely on any trusted third-party to mix the coins. (See Figure 18.4)
However, Zerocoin has some drawbacks, such as large overhead, limited functionality, and vulnerability to attacks.

18.2.2 Zcash

One of the most efficient cryptographic methods to create zero-knowledge proofs is zk-SNARK, which stands for "zero-knowledge, succinct and non-interactive arguments of knowledge". A zk-SNARK proof is short and easy to verify.

Zcash is a cryptocurrency that enhances the privacy of Bitcoin transactions by using the UTXO framework and the zk-SNARK cryptographic tool. Zcash introduces new types of transactions that hide the origin, destination, and amount of the payment. These transactions also allow the coins to be split and merged.

Zcash uses zk-SNARK to create efficient proofs that replace the traditional links between inputs and outputs. The proof shows that :

- the sender has the secret keys that match some of the public keys of previous outputs (the set of outputs is hidden)
- the sender does not spend more than the total amount of these outputs.

This helps to conceal both the identity and the value of the coins transferred.

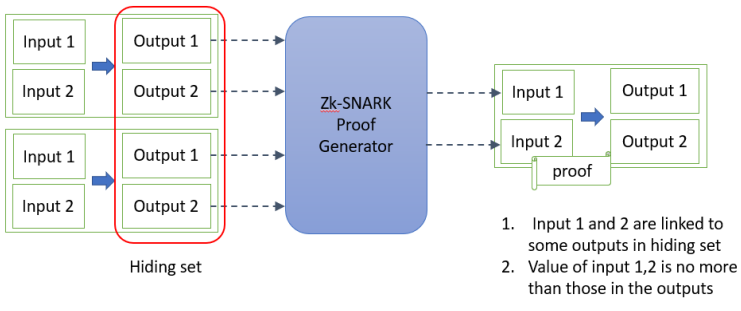


Figure 18.5: Zcash

18.3 Zero-knowledge Model

18.3.1 Language and NP

Definition 18.3.1: Language

A language L is a set of statements such that

$$L(x) = 1 \qquad \text{if} \quad x \in L$$

Language is a way of defining a set of statements that have a certain property. For example, the language of composite numbers is the set of statements that say that a number is composite, which means that it has more than two factors.

Definition 18.3.2: NP

We define NP to be the class of languages L that have a polynomial time Verifier V, such that

$$L(x) = 1 \Leftrightarrow \exists w, \quad \text{s.t.} \quad V(x, w) = 1$$

NP is a way of defining a class of languages that have a certain property. For example, the class of languages that can be verified in polynomial time by a verifier. A verifier is an algorithm that takes two inputs: a statement and a witness. A witness is an additional piece of information that helps to prove the statement.

Example 18.3.1 (NP)

x is an integer, L is testing for composite. We can
nd a witness here to be the prime factorization of x, and the veri
er can testify whether the product of w equals to x in polynomial time.

18.3.2 Prover and Zero-knowledge

Prover and Zero-knowledge are two roles that are involved in a zero-knowledge proof.
For example, if the statement is that a number x is composite, then the Prover is the one who knows the factorization of x, and the Verifier is the one who wants to check if the statement is true.The Prover can send x and its factors w to the Verifier, and claim that x is composite. The Verifier can then ask the Prover to prove it by using some mathematical technique that does not reveal w. This way, the Verifier can be convinced that x is composite, but cannot learn anything about w

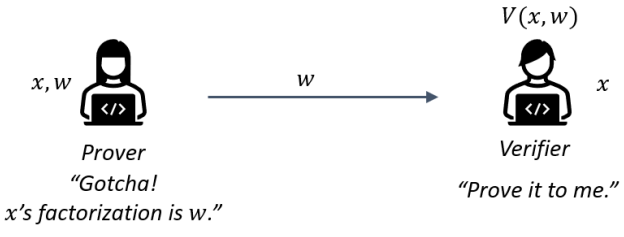


Figure 18.6: Prover and Verifier

A zero-knowledge proof system has three security requirements:

- **Completeness**: This means that if the statement is true and the prover knows a valid witness, then the prover can always persuade the verifier to accept the proof.
- **Soundness**: This means that if the statement is false ($x \notin L$), then the verifier cannot be fooled by any prover to accept the proof.

- **Zero-knowledge**: This means that the proof does not reveal any information about the witness or any other secret information to the verifier.

Imagine that you have a number x and you want to prove that you know another number w that satisfies the equation $g^w = x$, where g is a special number that can generate a group of numbers with a prime size q. To achieve zero-knowledge, the verifier only has access to x and should learn nothing about w during the verification process. The proof is done by following these steps:

1. The prover picks a random $v \in \mathbb{Z}_q^*$, and sends the verifier $t = g^v$.
2. The verifier chooses a random number $c \in \mathbb{Z}_q^*$, and sends it back to the prover.
3. The prover computes $r = v - cw$ and returns r to the verifier.
4. Finally, the verifier checks if $t = g^r x^c$

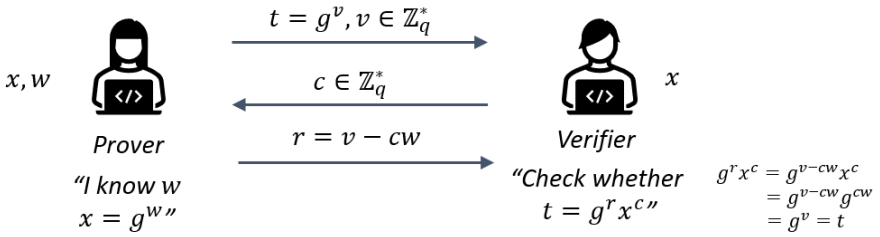


Figure 18.7: Zero-knowledge proof

As we can see, the proof and verification process do not reveal anything about w, because it is hidden by random numbers. This is why it is called "zero knowledge". The amazing thing is that every problem in NP has a zero knowledge proof.

18.3.3 Efficiency

The efficiency of the cryptographic tools (e.g., how much time they need to create the proof or check the proof, how big the proof is) is important for practical use. We use T to represent the time it takes to run V(x,w), which is the basic complexity of verification. We look at four measures of complexity of zero knowledge proof systems :

- **Prover complexity** : Provers who are efficient can make proofs in an expected time of $O(T \log T)$. This is only slightly more than the basic complexity.
- **Verification complexity and proof size.** : A good feature of these proof systems is that they are succinct, which means that the proof size is small and can be checked quickly. The proof size can be constant or logarithmic compared to the statement size. The proof can be verified in an expected time of O(1) or O(log T), which means that it does not depend on the statement size. However, the verification is still not very efficient for practical use (for example, in Zcash), because the constants are large.
- **Interactivity of verification** : Many zero knowledge protocols require interaction between the prover and the verifier. However, noninteractive proofs are more suitable for blockchain applications. There is a general technique to make interactive protocols noninteractive while keeping their security properties (this is called the Fiat-Shamir heuristic), but it works under ideal conditions (including needing a special model called random oracle) and it is not very practical. Therefore, we are interested in finding more specific approaches for noninteractive proofs.
- **Setup assumptions** : Some zero knowledge protocols need a "trusted setup", for example, zk-SNARKs. This means that a trusted party has to compute the parameters that are needed to make and check the proofs. If this party is not honest, the protocol could be broken and the money could be created out of thin air. To avoid this, some protocols like zk-STARKs (Zero-Knowledge Scalable Transparent ARguments of Knowledge) use public randomness instead of trusted setup to create systems that can be verified without trusting anyone. These systems are based on advanced cryptography and are being developed into practical libraries by researchers.

18.4 From Zero Knowledge Proof Systems to Zcash

We create new data structures and ways of addressing on top of Bitcoin's design to enable anonymity in Bitcoin using zero knowledge proof systems.

- **Address** : Like Bitcoin, Zcash has two kinds of addresses: public key address $addr_{pk}$ and secret key address $addr_{sk}$.
- We define a coin c, which has the same role as a transaction output of Bitcoin, with the following attributes:
  - Coin commitment $cmt(c)$
  - Coin value $v(c)$
  - Coin serial number $sn(c)$
  - Coin address $addr_{pk}(c)$

  The coin commitment is a way of hiding the data inside the coin by using a special function that produces a unique output for each input. The serial number is a way of identifying the coin by using another function that produces a random-looking output for each input. Both functions are based on a mathematical operation called SHA256 compression, which is also used to create the addresses and the coin attributes.
- **Pour transaction structure** : A pour transaction is a type of transaction that has two inputs and two outputs. Unlike Bitcoin transactions, the pour transaction only shows the serial numbers of the coins that are used as inputs, but it hides everything else, such as how much money the coins are worth, or who owns them.

  All the publicly visible information of the transaction can be denoted as :

$$tx := (rt; sn_1^{old}; sn_2^{old}; cmt_1^{new}; cmt_2^{new}; v_{pub}; \text{info}; \text{proof})$$

  which rt is the Merkle root; inputs, which are the serial numbers of the old coins; outputs, which are the commitments of the new coins; v_{pub}, which is the part of the input value that can be optionally revealed to the public; info, which is a transaction string that can be optionally added; and proof, which is used to show that the old coins belong to the sender and that the transaction is valid. The pour transaction allows both splitting and merging of the coins by using two inputs and outputs. This means that multiple coins can be split or merged by using multiple pour transactions. We will explain these properties in more detail below.

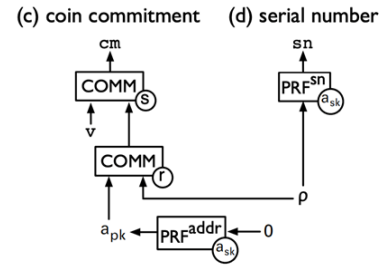


Figure 18.8: Data structure

18.5 Zcash Framework

Our goal is to design a pour transaction that hides the details of the coins (especially the public keys) that are involved in creating two new coins from two old coins. This is the transaction linkage problem that Bitcoin's UTXO state management system has to deal with.

18.5.1 First Attempt: use commitment

First We try to construct the transaction using only the coin commitments, i.e. a pour transaction has $(cmt_1^{old}, cmt_2^{old}, cmt_1^{new}, cmt_2^{new}, \text{proof})$ as its components.

The problem formulation can be expressed as an NP statement, abstracting away the details of how the zero knowledge proof is generated. The statement includes:

- $x = (cmt_1^{old}, cmt_2^{old}, cmt_1^{new}, cmt_2^{new})$
- $L(x)$ is an indicator of whether x is a valid pour transaction. $L(x) = 1$ if x is a valid pour transaction.
- $w = \left(c_1^{old}, c_2^{old}, c_1^{new}, c_2^{new}, addr_{sk}(c_1^{old}), addr_{sk}(c_2^{old})\right)$

Given x and w, a verifier $V(x, w) = 1$ if the following conditions are true:

- The commitments of four coins are correct
- $v(c_1^{old}) + v(c_2^{old}) \geq v(c_1^{new}) + v(c_2^{new})$
- $addr_{sk}$ of two old coins matches $addr_{pk}$

It is easy to see that $L(x) = 1 \Leftrightarrow V(x, w) = 1$.

We use algebraic circuits to represent these statements and generate the proof by **circuit satisfiability**. The proof is an encrypted pair $\pi = [g^H, g^Z]$, where H and Z are polynomials that are calculated during the proving phase, and there is a verification key $vk = g^T$. We saw in the discrete logarithm example above that the proving process did not expose the witness information because it is hard to invert the logarithm. You can read more about an example construction in this post.

Now anyone in the blockchain who has access to w can produce the proof without revealing any information related to w, and anyone who has access to x can check the validity of the transaction. But this method has a weakness that the commitment can still be traced.

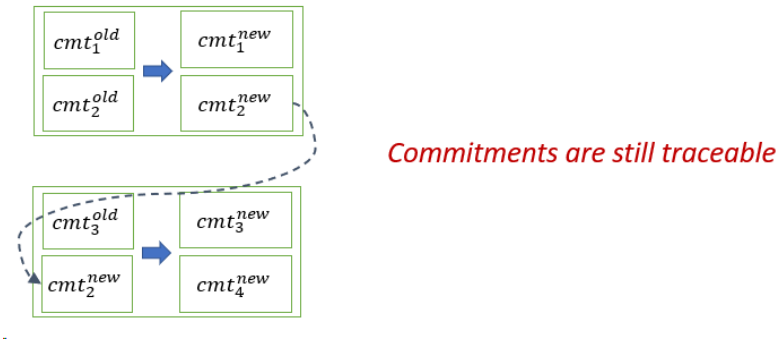


Figure 18.9: First attempt: Commitment

18.5.2 Second Attempt: two commitments

A possible improvement on the previous approach is to use two kinds of commitments:

1. a regular commitment cmt
2. a unique serial number sn (created by a pseudo-random generator)

In a transaction, we use serial numbers to represent the old coins and commitments to represent the new coins, i.e., the transaction contains $(rt; sn_1^{old}, sn_2^{old}, cmt_1^{new}, cmt_2^{new}, \text{proof})$, where rt indicates the Merkle-tree root of the commitments of the outputs in the ledger. The witness remains the same,

$$w = \left(c_1^{old}, c_2^{old}, c_1^{new}, c_2^{new}, addr_{sk}(c_1^{old}), addr_{sk}(c_2^{old})\right)$$

And again $L(x) = 1$ if and only if $V(x, w) = 1$, $V(x, w) = 1$ if the following conditions hold:
For $i \in \{1, 2\}$:

- The commitments cmt_i of c_i^{new} appear on the ledger, verified using the Merkle-tree root rt, i.e.

$$cmt(c_1^{new}) \in rt \wedge cmt(c_2^{new}) \wedge rt$$

- The address secret key $addr_{sk}^{old}{}_{,i}$ matches the address public key of c_i^{old}.
- $\left(sn(c_1^{old}), sn(c_2^{old}), cmt(c_1^{new}), cmt(c_2^{new})\right) = \left(sn_1^{old}, sn_2^{old}, cmt_1^{new}, cmt_2^{new}\right)$
- $v(c_1^{old}) + v(c_2^{old}) \geq v(c_1^{new}) + v(c_2^{new})$

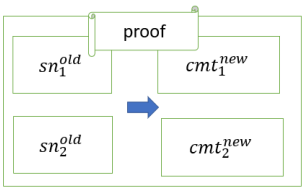


Figure 18.10: Second Attempt: two commitments

This formulation is similar to the previous one, in that those who possess w can create the proof and those who possess x can verify the validity. However, unlike the previous method, there is no direct link between the old coins and the new coins since they use different kinds of commitments. The only issue is that without the link, how can we tell which previous output is being spent? To address this issue, we keep track of all serial numbers that appear in previous transactions as a nullifier set and perform an extra check to see if the input serial number is already in the nullifier set.

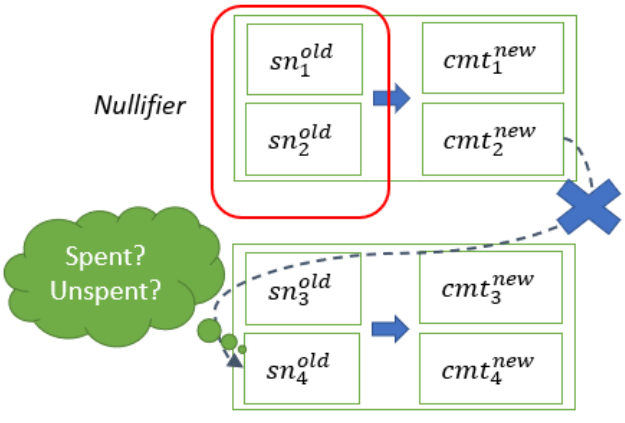


Figure 18.11: Check Unspent Coins

18.6 Zcash Protocol: Putting it all together

Zcash modifies Bitcoin by introducing pour transactions. Unlike a normal UTXO transaction, the pour transaction uses commitments and serial numbers instead of inputs and outputs to hide the connection between the old and new coins.

18.6.1 Create a pour transaction

The complete structure of a pour transaction, as we defined earlier, consists of

$$(rt; sn_1^{old}; sn_2^{old}; cmt_1^{new}; cmt_2^{new}; v_{pub}; \text{info}; \text{proof})$$

To make a pour transaction, the sender needs the following information:

- Old coins c_1^{old} , c_2^{old}
- Secret keys of old coins $addr_{sk}(c_1^{old})$, $addr_{sk}(c_2^{old})$
- New values v_1^{new} , v_2^{new}
- Public value v_{pub} s.t. $v_1^{old} + v_2^{old} \geq v_1^{new} + v_2^{new} + v_{pub}$
- New addresses $addr_{sk}(c_1^{new})$, $addr_{sk}(c_2^{new})$

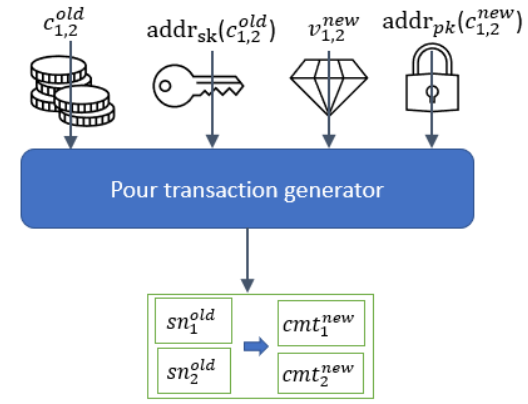


Figure 18.12: Generate a pour transaction

A real system can use a transaction generator to create a pour transaction and the new coins for the sender, given the necessary information. The sender then sends the new coins to the recipients off chain and the transaction is broadcasted on chain.

18.6.2 Generate a zk-SNARK proof

zk-SNARK is a cryptographic method that allows zero knowledge, succinct and non-interactive verification. It means that a prover who has a witness for an NP-statement can generate a short proof that anyone can verify without revealing the witness. The NP-statements in Zcash are usually statements of satisfiability, such as "the input matches this specific value for this specific hash function".

There are three polynomial time algorithms involved in the proving process, which are part of a library.

- **KeyGen** : KeyGen is the trusted setup that creates the proving and verifying keys, pk and vk, once for all.
- **Prove** : The function Prove takes pk, the witness w and the public input x and produces a short proof $\pi = Prove(pk, w, x)$.
- **Verify** : The function Verify takes vk, the public input x and the proof π and outputs a Boolean value $Verify(vk, x, \pi)$.

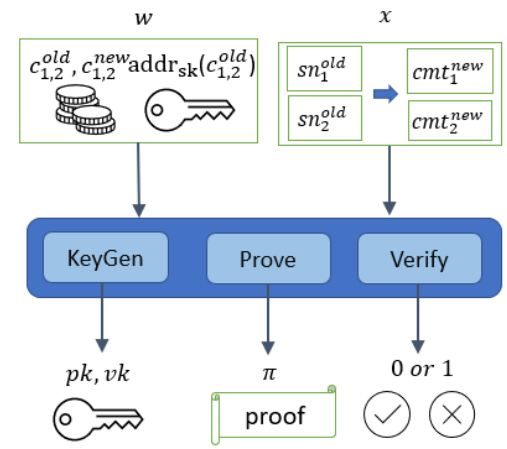


Figure 18.13: Generate a zk-SNARK proof

18.6.3 Incentives in Zcash

Zcash is a privacy-enhanced version of Bitcoin that uses the same basic protocol with an added feature of hiding transaction details. The incentives in Zcash are the same as in the Bitcoin protocol, which include both mining rewards and transaction fees. Since the zk proof verification process is fast, the verification time does not increase much (especially compared to the slow mining rate in Bitcoin).

18.7 Privacy

Privacy refers to how well the transactions hide the details of the sender, receiver, and amount. Programmability refers to how flexible the transactions support smart contracts, which are self-executing agreements with complex rules and logic.

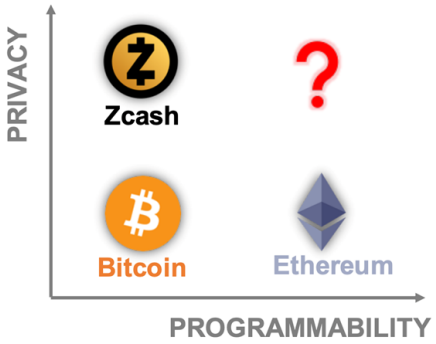


Figure 18.14: Cryptocurrencies in term of privacy and Programmability

Figure 18.14 shows that Zcash has high privacy and low programmability, Bitcoin has low privacy and low programmability, and Ethereum has low privacy and high programmability.