Chapter 14

Bitcoin and Nakamoto's Blockchain Protocol

Back in the 1970s, the study of distributed consensus was motivated by the need to build reliable aircraft control systems replicated on multiple computers [WLG+89]. Later on, distributed consensus protocols became widely deployed in companies such as Google and Facebook. These companies need to replicate their mission-critical infrastructure such as Google Wallet or Facebook Credit, and thus the challenge of achieving consistency naturally arises. In all of these classical scenarios, consensus is typically deployed on a *small scale*, involving three to dozens of machines. Participation is *closed*, i.e., only a preconfigured, known set of nodes can join the protocol — such environments are often referred to as *permissioned* environments.

Bitcoin [Nak08] came around in 2009 and gained popularity rapidly. As the Wikipedia page explains it [wik]:

> "Bitcoin is a cryptocurrency. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries. ... Bitcoin was invented in 2008 by an unknown person or group of people using the name Satoshi Nakamoto [Nak08] and started in 2009 when its source code was released as open-source software".

At the core of Bitcoin is a *blockchain* protocol (defined in Chapter 6) that allows a set of distributed nodes to agree on an ever-growing, linearly-ordered log of transactions. In fact, the term "blockchain" was popularized due to Bitcoin.

Bitcoin is not just an empirical success, it is also a scientific breakthrough! Specifically, Bitcoin's blockchain protocol, often called Nakamoto's

blockchain [Nak08], is the first to demonstrate the feasibility of reaching consensus in a *permissionless* environment. In a permissionless environment, anyone is free to join the consensus protocol at any time. Since there is no a-priori knowledge of the identities of the participants, participants must communicate through *unauthenticated channels*.

To reach consensus in such a permissionless environment, one big challenge is the so-called "Sybil attack". Since the communication channel is unauthenticated, anyone can impersonate anyone else; and a single machine can also impersonate many machines, e.g., in an attempt to outnumber the honest players and disrupt the consensus. Exactly because of this reason, the classical insight had always been that consensus is impossible in such a permissionless environment without even authenticated communication channels. Indeed, with some effort, one can formalize this intuition and mathematically prove that absent any other assumptions, consensus is impossible in such a permissionless environment [PS17b].

Of course, the mathematical impossibility did not stop Bitcoin. Nakamoto's blockchain protocol circumvented this impossibility by leveraging Proof-of-Work (PoW). The idea is that players need to solve computational puzzles to cast votes. Roughly speaking, a player's voting power is proportional to its computational power. Moreover, the blockchain protocol guarantees consistency and liveness as long as *the majority of the mining power in the system is honest*.

In this lecture, we will describe how Nakamoto's blockchain works, and prove its security.

14.1 Nakamoto's Ingenious Idea in a Nutshell

Block format and notations. In Nakamoto's protocol, each honest node maintains a blockchain denoted chain at any point of time. The first block in the blockchain, denoted $\mathsf{chain}[0]$, is a canonical block called the *genesis*. Every other block $\mathsf{chain}[i]$ where $i > 0$ is of the format $\mathsf{chain}[i] := (h_{-1}, \eta, \mathsf{txs}, h)$, containing the hash of the previous block denoted h_{-1}, a puzzle solution η, a payload string txs which may contain a set of transactions to be confirmed, and a hash h of the present block. We will use the notation:

- We use $\mathsf{chain}[-\ell]$ to denote the ℓ-th to last block in chain. For example $\mathsf{chain}[-1]$ denotes the last block and $\mathsf{chain}[-2]$ denotes the second to last block, and so on;
- We use $\mathsf{chain}[:\ell]$ to denote the prefix $\mathsf{chain}[0..\ell]$.

- We use $\mathsf{chain}[:-\ell]$ to denote the prefix of chain except for the last ℓ blocks.
- We use $|\mathsf{chain}|$ to denote the length of chain, i.e., the total number of non-genesis blocks in chain.
- We often use the notation "$\_$" to denote a wildcard field that we do not care about.

Remark 21 (Bitcoin's genesis block)**.** Bitcoin's genesis block embedded the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". According to Bitcoin's Wiki page [gen], "this was probably intended as proof that the block was created on or after January 3, 2009, as well as a comment on the instability caused by fractional-reserve banking.".

Mining. Given some blockchain chain, let its last block be $(\_, \_, \_, h^*)$. To "mine" a new block off chain, let txs be the outstanding transactions — a miner would try random puzzle solutions $\eta \in \{0,1\}^\lambda$ and check if

$$\mathsf{H}(h^*, \eta, \mathsf{txs}) < D_p$$

where H denotes a Proof-of-Work (PoW) oracle (implemented as a hash function), D_p is some appropriate difficulty parameter. In Bitcoin, D_p is chosen such that in expectation it takes all miners combined 10 minutes to mine a new block. We will elaborate on how to choose D_p later in Section 14.3. If some puzzle solution η produces a hash outcome that is smaller than D_p, then the tuple $(h^*, \eta, \mathsf{txs}, \mathsf{H}(h^*, \eta, \mathsf{txs}))$ forms a valid block extending from chain; and chain is often said to be the parent chain of the newly mined block. Nodes propagate any new block they have mined.

Roughly speaking, we assume that the PoW function H behaves like a random function, and there is no algebraic shortcut one can exploit when evaluating H. In other words, there is no better way to find puzzle solutions than brute-force trying many different solutions. This is why mining is a computationally expensive process.

Because each block contains a hash of the previous block, the entire chain is bound together by the cryptographic hash. In other words, assuming that no hash collisions are found, then a block uniquely binds to its entire prefix.

Longest chain. One of the most beautiful ideas in Nakamoto's construction is the longest chain idea. Miners always try to mine a block off the *longest chain* it has seen. At any time, all but the last K blocks in the longest chain are considered *final*. In other words, if a transaction tx is embedded K blocks deep in the present longest chain (i.e., at least K blocks away from

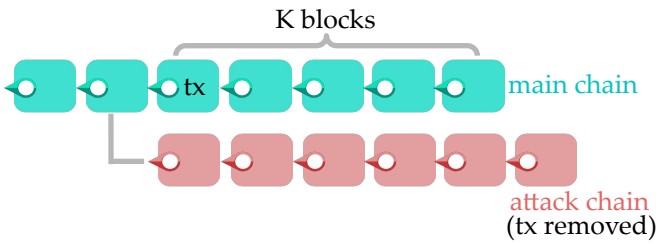

Figure 14.1: The adversary aims to undo transaction tx by mining a longer fork than the main chain. If the adversary has only minority of the mining power, statistically speaking, it is extremely unlikely that the adversary can win the race against the main chain. Let K be how deep tx is embedded in the main chain: as K increases, our confidence in tx's finality increases very sharply.

the end), we may treat the transaction as finalized. Moreover, the larger the K, the more confident we are about tx's finality.

To intuitively understand why, it helps to look at Figure 14.1. Suppose tx is contained in the block $\mathsf{chain}[-K]$ where chain denotes the longest chain observed thus far. Imagine that tx corresponds to the payment the adversary made to a Ferrari dealer to purchase a Ferrari. Once the car has been shipped to the adversary, the adversary may want to undo tx and reverse its payment. Can the adversary succeed in such an attack?

Informally, to undo tx, the adversary would have to mine an attack fork off some prefix $\mathsf{chain}' \preceq \mathsf{chain}[:-K]$; not only so, the attack fork must be longer than the main chain for it to win — but keep in mind that the main chain is growing too. Thus the adversary must, within a fixed time window, mine at least K more blocks than the honest nodes, to win this race. If the adversary controls only minority of the mining power, it is statistically unlikely that it can succeed; and further, the larger the K, the exponentially smaller the adversary's chance of success!

The above is not a formal proof why Nakamoto's blockchain preserves consistency, but we will formally prove it in Chapter 17.

14.2 Nakamoto's Blockchain: Formal Description

We will formally describe a stripped-down version of the full Nakamoto consensus protocol implemented in Bitcoin. One simplification we make is

to pretend that the total mining power in the system is known and fixed. In the Bitcoin's implementation, this assumption is not true, and therefore the protocol relies on a difficulty adjustment mechanism to adjust the difficulty of the computational puzzles being solved based on how much mining power is present in the recent past. Our description will omit this difficulty adjustment mechanism, and the resulting simplified protocol is often called the "barebone" Nakamoto's blockchain.

We will assume a synchronous network where honest nodes' messages must be delivered within at most Δ delay to honest recipients (see Chapter 6).

Modeling PoW puzzles. We will use $(\mathsf{H}, \mathsf{H.ver})$ to denote a PoW scheme where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ is the PoW's work function; and $\mathsf{H.ver} : \{0,1\}^* \to \{0,1\}^\lambda$ is the corresponding verification function. Recall that H requires the caller to expend work and evaluate a hash function, and $\mathsf{H.ver}$ is used to check if a purported puzzle solution is correct.

Without loss of generality, we may assume that all nodes have equal computational power and we use n to denote the total number of nodes — if a node has more computational power, it can be viewed as multiple nodes. This way, saying that "the majority of nodes are honest" equates to saying that "the majority of the computational power is honest". Every node can only query the PoW function H at a bounded rate. We may assume that in every round, each node can invoke H at most once — this is without loss of generality since we can always rename the time it takes to evaluate H as one round. However, we do not impose any limit on calls to the verification function $\mathsf{H.ver}$.

Remark 22. In Nakamoto's protocol, $\mathsf{H.ver}$ is only called when messages (specifically, blocks) are received from the network. In practice, since the network has limited bandwidth, $\mathsf{H.ver}$ is called significantly fewer times than H (even when an adversary may flood the network with fake blocks). This is why we do not charge calls to $\mathsf{H.ver}$.

Barebone Nakamoto's blockchain. Nodes always try to mine blocks off the *longest* valid chain they have observed thus far. Once a block is mined, the miner propagates it to others. At any time, the longest chain a node has observed with the last K blocks removed is considered as the current finalized log. We now describe the protocol more formally.

Although not explicitly noted below, we make an *implicit echoing* assumption: whenever a node hears a fresh message from the network previously

unseen or receives a new transaction as input, it echos the message or transaction to everyone else. This assumption makes sure that if any honest node sees a message in round r, then all honest nodes will have observed it by round $r + \Delta$.

Nakamoto's blockchain

- Nodes that are newly spawned start with initial chain containing only a special genesis block: $\mathsf{chain} := (0, 0, \bot, \mathsf{H}(0, 0, \bot))$.
- Whenever a node hears a message chain' from the network, if incoming message chain' is a valid blockchain and it is longer than its current local blockchain chain, replace chain by chain'. We define what it means for a chain to be valid later. Checking the validity of chain' can be done using only $\mathsf{H.ver}$ queries.
- In every round, try to mine a new block off the longest chain seen so far (denoted chain) as follows. Let txs be the outstanding transactions observed so far that are not contained in the current chain. Now parse $\mathsf{chain}[-1] := (\_, \_, \_, h_{-1})$, pick a random solution $\eta \in \{0,1\}^\lambda$, and issue query $h = \mathsf{H}(h_{-1}, \eta, \mathsf{txs})$. If $h < D_p$, then append the *newly mined* block $(h_{-1}, \eta, \mathsf{txs}, h)$ to chain and send $\mathsf{chain}||(h_{-1}, \eta, \mathsf{txs}, h)$ to everyone. The parameter D_p determines how difficult it is to mine a block, and how to choose D_p will be explained in Section 14.3 below.
- At any time, a node's *finalized log* is defined to be $\mathsf{chain}[: -K]$, i.e., the longest chain observed so far removing the last K blocks. We need to choose K to be sufficiently large such that the probability of breaking consistency is extremely small (see Theorem 17 for a rigorous statement).

Valid chain. We say a block $\mathsf{chain}[i] = (h_{-1}, \eta, \mathsf{txs}, h)$ is *valid with respect to a predecessor block* $\mathsf{chain}[i-1] = (h'_{-1}, \_, \_, h')$ if the following conditions hold: $h_{-1} = h'$, $h = \mathsf{H}(h_{-1}, \eta, \mathsf{txs})$, and $h < D_p$. A chain of blocks chain is *valid* iff:

1. $\mathsf{chain}[0] = (0, 0, \bot, \mathsf{H}(0, 0, \bot))$ is the genesis block, and
2. for all $i \in [\ell]$ where $\ell := |\mathsf{chain}|$, $\mathsf{chain}[i]$ is valid with respect to $\mathsf{chain}[i-1]$.

14.3 Choosing the Mining Difficulty Parameter

How should we choose the mining difficulty parameter? In Bitcoin, the difficulty parameter is chosen such that on average, all miners combined take 10 minutes to mine the next block. Of course, 10 minutes seem awfully long, especially given that one also has to wait for a transaction to be embedded K blocks deep for it to be confirmed. In practice, many consider $K = 6$ to be secure enough — this means it could easily take *an hour* for a transaction to confirm! From a confirmation delay perspective, it seems desirable to make the puzzles less difficult such that blocks are confirmed more frequently — but would this be safe?

It turns out that we cannot arbitrarily lower the puzzles' difficulty; doing so could break the consistency of the consensus protocol. One way to think of the matter is the following: since the network delay can be up to Δ, whenever a new block is mined, there is a Δ gap in which the new block is being propagated on the network to the honest nodes, and during this Δ gap, the honest nodes are not doing any useful work[1]! On the other hand, the adversary may not need to suffer from the same Δ delay (e.g., the adversary controls a mining farm where blocks are transmitted over dedicated links). This gives the adversary an advantage when it tries to mine an attack fork like in Figure 14.1. One can think of the advantage in terms of the honest mining power that is *discounted* by the network's delay Δ.

To understand how much honest mining power is discounted by Δ, we give an *informal* back-of-the-envelope calculation — this calculation is only to convey intuition and it should *not* be interpretted as a formal proof. Let p be the probability that a single node mines a block in any fixed round. Suppose that there are n nodes, 51% of which are honest. The probability that the honest nodes combined can mine a block in a round is roughly $1 - (1-p)^{0.51n} \approx 0.51pn \ll 1$. The expected number of rounds till a new honest block is mined is roughly $\frac{1}{0.51pn}$. Now, it takes Δ rounds to propagate the block. Suppose that all of the honest nodes' work is wasted during the Δ rounds, then roughly speaking, every $\frac{1}{0.51pn}$ rounds, we end up wasting Δ rounds. In this sense the discount ratio is roughly

$$\frac{\frac{1}{0.51pn}}{\frac{1}{0.51pn} + \Delta} = \frac{1}{1 + 0.51pn\Delta} \approx 1 - 0.51pn\Delta$$

[1] Let us ignore the tiny probability that honest nodes mine two consecutive blocks during the Δ interval.

Exercise 31. With the above back-of-the-envelope calculation, the term $(1 - 0.51pn\Delta) \cdot (0.51n)$ can be regarded as the *discounted honest mining power*. Now, suppose that $0.49n > (1 + \epsilon) \cdot (1 - 0.51pn\Delta) \cdot (0.51n)$, i.e., the corrupt mining power exceeds the discounted honest mining power by some constant $\epsilon \in (0, 1)$ margin. Describe how the adversary can succeed in mining an attack fork like in Figure 14.1 with probability almost 1, despite the fact that it controls only 49% of the mining power.

The above exercise suggests that for Nakamoto's consensus protocol to maintain consistency, more precisely speaking, we need not just honest majority in mining power, but a slightly more stringent condition, that is,

> *the honest mining power, even when discounted by the network delay Δ, must exceed the corrupt mining power!*

Setting the puzzles to be more difficult makes the discount factor smaller.

Formal requirements on the mining difficulty. We will formally articulate a set of requirements on the mining difficulty — under this set of parameters, we shall be able to formally state and prove the security properties of Nakamoto's blockchain.

Our goal is to set the D_p parameter, called the mining difficulty parameter, in the protocol formally described in Section 14.2. D_p can be chosen in the following way:

- first, choose an appropriate probability $p \in (0, 1)$ as described below;
- once p is fixed, we choose D_p such that the probability that any player mines a block in a round is $p \in (0, 1)$. This can be achieved[2] by setting $D_p := p \cdot 2^\lambda$ such that for all (h, txs), $\Pr_\eta[\mathsf{H}(h, \eta, \mathsf{txs}) < D_p] = p$.

We choose the parameter $p \in (0, 1)$ such that it satisfies the following conditions where n is the total number of nodes, and Δ denotes the maximum network delay:

1. $\nu := 2pn\Delta < 0.5$; and
2. Honest mining power, even when discounted by the network delay, must outnumber corrupt mining power by an appropriate constant margin.

[2]For simplicity, we shall assume that $p \cdot 2^\lambda$ is an integral number.

Formally, let $\phi \in (0,1)$ be an arbitrarily small constant, and let ρ denote the fraction of corrupt nodes. We require that

$$\frac{1-\rho}{\rho} \geq \frac{1+\phi}{1-\nu} \tag{14.1}$$

Note that the second requirement can be equivalently interpreted as $(1-\rho)\cdot(1-\nu) \geq (1+\phi)\cdot\rho$, where $1-\rho$ is the fraction of honest mining power and ρ is the fraction of corrupt mining power. The term $1-\nu = 1-2pn\Delta$ is the discount in the honest mining power. The constant 2 here differs from our earlier back-of-the-envelope calculation — but it turns out that this is the constant needed for the formal proofs we present in Chapter 17[3].

If $\Delta = 0$, i.e., all messages are received instantly without any delay, then no discount would be incurred — in this case, Equation (14.1) simply boils down to requiring that the honest mining power exceed the corrupt mining power by an arbitrarily small constant margin $\phi \in (0,1)$. The larger the network delay Δ, the more disadvantageous it is to the honest nodes, and thus the more honest fraction we will need to ensure consistency.

14.4 Properties of Nakamoto's Blockchain

To state the formal guarantees attained by Nakamoto's blockchain, we shall assume that Nakamoto's blockchain protocol is executed for $\mathsf{poly}(\lambda)$ number of rounds where λ is the security parameter. This is a reasonable assumption for cryptographic protocols, where we typically assume that the adversary is polynomially bounded in the security parameter λ. We say that $\mathsf{negl}(\lambda)$ is a *negligible function* if for any fixed polynomial function $p(\lambda)$, there exists λ_0 such that for any $\lambda > \lambda_0$, $\mathsf{negl}(\lambda) < 1/p(\lambda)$. In other words, a negligible function is one that drops off very sharply as we increase the security parameter λ. If a protocol's failure probability is negligible in the security parameter λ, then by increasing the security parameter λ a little, we can make the failure probability extremely small.

Nakamoto's blockchain, when instantiated with appropriate parameters stated in Section 14.3, satisfies the following theorem, which we shall prove in Chapter 17. Below we will state the theorem formally first, and then we will give intuitive explanations for each property.

Theorem 17. *Suppose that $K = \omega(\log \lambda)$ and let $\epsilon \in (0,1)$ be an arbitrarily small constants. There exists a negligible function $\mathsf{negl}(\cdot)$, such that with $1-$*

[3]It is possible that the constant 2 can be tightened with tighter proofs, but doing so is beyond the scope of this course.

$\mathsf{negl}(\lambda)$ *probability over the choice of the randomized execution of Nakamoto's blockchain, the following properties hold:*

- **Chain growth lower bound.** Let $\alpha := (1-\rho)np$ denote the expected number of honest nodes that mine a block in each round. For any round r_0 and any duration $t \geq \frac{K}{\alpha}$, let chain^{r_0} be some honest node's longest chain in round r_0 and let chain^{r_0+t} be some honest node's longest chain in round $r_0 + t$ (the two honest nodes can be the same or different). It must be that

$$|\mathsf{chain}^{r_0+t}| - |\mathsf{chain}^{r_0}| \geq (1-\epsilon)(1-2pn\Delta)\alpha t$$

- **Chain quality.** Let chain be the longest chain of some honest node sometime during the protocol execution: it must be that for any K consecutive blocks $\mathsf{chain}[j..j+K]$ in this longest chain, more than $\mu := 1 - \frac{1+\epsilon}{1+\phi}$ fraction of the blocks are mined by honest nodes.

- **Consistency.** Let chain^r denote some honest node's longest chain in round r and let chain^t denote some honest node's longest chain in round $t \geq r$ (note that the two honest nodes can be the same or different). It must hold that

$$\mathsf{chain}^r[:-K] \preceq \mathsf{chain}^t$$

  where $\mathsf{chain} \preceq \mathsf{chain}'$ means that the former is a prefix of the latter or they are the same chain.

Below we elaborate on these properties and provide some intuition for each of them.

14.4.1 Chain Growth Lower Bound

Intuitively, chain growth lower bound says that honest nodes' chains must grow steadily over time. Of course, the corrupt nodes could completely stop mining, and therefore we can only guarantee that honest nodes' chains grow at a rate proportional to the honest nodes' total mining power which is α. However, keep in mind that the network has maximum delay Δ, and every time a block is mined, Δ rounds can be wasted just transmitting the block to others. For this reason, the actual chain growth rate we can guarantee is only $(1-2pn\Delta)\alpha$, where the honest mining power α is further discounted by the factor $1-2pn\Delta$.

Why do we care about chain growth? Because chain growth is necessary for achieving liveness (see Section 6), i.e., transactions submitted must be

included in honest nodes' finalized logs fairly soon. It is not hard to see why chain growth is necessary for liveness, but it turns out that chain growth alone is not sufficient for ensuring liveness. For example, it could be that although the blockchain grows, every block is mined by corrupt players, and corrupt players may not include outstanding transactions in their mined blocks or they may selective drop certain transactions. For this reason, we also need chain quality which ensures that every now and then, some block mined by honest nodes makes its way into the blockchain.

14.4.2 Chain Quality

Chain quality says that in every window of consecutive K blocks in honest nodes' longest chains, it must be that more than $\mu := 1 - \frac{1+\epsilon}{1+\phi}$ fraction of them are mined by honest nodes. Chain quality is necessary for ensuring liveness, that is, transactions submitted must be included in honest nodes' finalized logs fairly soon. If (non-zero) chain quality holds, intuitively, it means that every now and then, an honest block makes its way into the blockchain. Since honest miners always include all outstanding transactions in the blocks they mine, liveness can be ensured (see also Exercise 33).

Does Nakamoto's blockchain achieve ideal chain quality? Although liveness only needs non-zero chain quality, it is natural to ask if the protocol provides *fairness*. If Nakamoto's protocol were completely "fair", the fraction of honest blocks ought to be $1 - \rho$. Henceforth the expression $1 - \rho$ is referred to as "ideal chain quality". Does Nakamoto's protocol provide ideal chain quality?

To understand this, let us try to gain some intuition about the chain quality parameter μ in Theorem 17. For simplicity, let us assume that $\Delta = 0$, and moreover, equality is taken in Equation (14.1). In this case, we simply have that

$$\frac{1-\rho}{\rho} = 1 + \phi$$

Since we can take $\epsilon \in (0, 1)$ to be very small, for a back-of-the-envelope calculation, we simply ignore ϵ. In this case, the chain quality parameter in Theorem 17 would roughly be

$$\mu \approx 1 - \frac{1}{1+\phi} = \frac{1-2\rho}{1-\rho}$$

For example,

- Suppose that the fraction of honest mining power $1 - \rho = 2/3$. Then, the chain quality μ guaranteed by Theorem 17 is roughly $1/2$, whereas ideal chain quality would be $2/3$.
- Suppose that the fraction of honest mining power $1 - \rho$ is sightly greater than $1/2$. Then, the chain quality μ guaranteed by Theorem 17 is slightly greater than 0, whereas ideal chain quality ought to be $1/2$.

  Is this mismatch due to looseness of the theorem, or is it that Nakamoto's blockchain is inherently not fair?

It turns out that Nakamoto's blockchain is *not fair*! A well-known attack, called the *selfish mining* attack [mtg, ES14], shows that if a coalition with roughly $\rho < 1/2$ fraction of mining power deviates from the honest protocol, it can, in the best-case scenario, control roughly $\rho/(1-\rho)$ fraction of the blocks! We will further explain the selfish mining attack in Chapter 15.

14.4.3 Consistency

The consistency property in Theorem 17 is stated w.r.t. nodes' longest chains, but not w.r.t. nodes' finalized logs. Recall that in Nakamoto's consensus, at any time, a node's finalized log is its longest chain but chopping off the trailing K blocks. If we want to prove that Nakamoto's blockchain realizes the blockchain abstraction defined earlier in Chapter 6, we need to prove the consistency property defined in Chapter 6, which is stated w.r.t. nodes' finalized logs. In other words, we want to prove that, with extremely high probability over the choice of the randomized execution, the following should hold: if LOG_i^r and LOG_j^t are the finalized logs of two honest nodes i and j in rounds r and t respectively, it must be that either $\mathsf{LOG}_i^r \preceq \mathsf{LOG}_j^t$ or $\mathsf{LOG}_i^r \succeq \mathsf{LOG}_j^t$. We leave this as a homework exercise.

Exercise 32. Recall that in the Nakamoto's blockchain, a node's finalized log is always its longest chain but removing the trailing K blocks. Suppose that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the consistency property stated in Theorem 17 is satisfied.

Prove that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the following holds: if LOG_i^r and LOG_j^t are the finalized logs of two honest nodes i and j in rounds r and t respectively, it must be that either $\mathsf{LOG}_i^r \preceq \mathsf{LOG}_j^t$ or

$\mathsf{LOG}_i^r \succeq \mathsf{LOG}_j^t$.

14.4.4 Liveness

In Exercise 32, we proved that Nakamoto's blockchain satisfies consistency as defined in Chapter 6. To prove that Nakamoto's protocol realizes a blockchain, we also need to show that it satisfies the liveness property as defined in Chapter 6.

Exercise 33. Suppose that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, chain growth lower bound and chain quality as stated in Theorem 17 are satisfied.

Prove that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the following holds: suppose that an honest node receives some transaction tx as part of its input in some round r, then, by round $r + \Theta(K/\alpha + \Delta)$, tx must appear in every honest node's finalized log.

Combining Exercise 32 and 33, we may conclude that Nakamoto's protocol indeeed satisfies the blockchain abstraction defined earlier in Chapter 6.