CHAPTER 2

How Bitcoin Achieves Decentralization

In this chapter, we discuss decentralization in Bitcoin. In Chapter 1, we looked at the crypto basics that underlie Bitcoin and ended with the description of a simple currency called Scroogecoin. Scroogecoin achieves a lot of what we want in a ledger-based cryptocurrency, but it has one glaring problem—it relies on a centralized authority (Scrooge). We ended with the question of how to decentralize, or de-Scrooge-ify, this currency. Answering that question is the focus of this chapter.

As you read through this chapter, note that the mechanism by which Bitcoin achieves decentralization is not purely technical—it is a combination of technical methods and clever incentive engineering. By the end of this chapter, you should have a really good appreciation for how this decentralization is achieved, and, more generally, how Bitcoin works and why it is secure.

2.1. CENTRALIZATION VERSUS DECENTRALIZATION

Decentralization is an important concept that is not unique to Bitcoin. The notion of competing paradigms of centralization versus decentralization arises in a variety of different digital technologies. To best understand how it plays out in Bitcoin, it is useful to understand the central conflict—the tension between these two paradigms—in a variety of other contexts.

On one hand we have the Internet, a famously decentralized system that has historically competed with and prevailed against “walled-garden” alternatives like AOL’s and CompuServe’s information services. Then there’s email, which at its core is a decentralized system based on the Simple Mail Transfer Protocol (SMTP), an open standard. Although it does have competition from proprietary messaging systems like Facebook or LinkedIn mail, email has managed to remain the default for person-to-person communications online. In the case of instant messaging and text messaging, we have a hybrid model that can't be categorically described as centralized or decentralized. Finally there's social networking: despite numerous concerted efforts by hobbyists, developers, and entrepreneurs to create alternatives to the dominant centralized model, centralized systems like Facebook and LinkedIn still dominate this space. In fact, this conflict long predates the digital era—we see a similar struggle between the two models in the history of telephony, radio, television, and film.

Decentralization is not all or nothing; almost no system is purely decentralized or purely centralized. For example, email is fundamentally a decentralized system based on a standardized protocol, SMTP, and anyone who wishes can operate an email server of their own. Yet what has happened in the market is that a small number of centralized webmail providers have become dominant. Similarly, even though the Bitcoin protocol is decentralized, services like Bitcoin exchanges, where you can convert bitcoins into other currencies, and wallet software (software that allows people to manage their bitcoins) may be centralized or decentralized to varying degrees.

With this in mind, let's break down the question of how the Bitcoin protocol achieves decentralization into five more specific questions:

1. Who maintains the ledger of transactions?
2. Who has authority over which transactions are valid?
3. Who creates new bitcoins?
4. Who determines how the rules of the system change?
5. How do bitcoins acquire exchange value?

The first three questions reflect the technical details of the Bitcoin protocol—these three questions are the focus of this chapter.

Different aspects of Bitcoin fall on different points on the centralization/decentralization spectrum. First, the peer-to-peer network is close to purely decentralized, since anybody can run a Bitcoin node, and the entry barrier is fairly low. You can go online and easily download a Bitcoin client and run a node on your laptop or your desktop. Currently there are several thousand such nodes. Second, Bitcoin *mining*, which we study in Section 2.4, is technically also open to anyone, but it requires a high capital cost. As a result, the Bitcoin mining ecosystem has a high degree of centralization or concentration of power. Many in the Bitcoin community see this as quite undesirable. Third, Bitcoin nodes run updates to the software, which has a bearing on how and when the rules of the system change. One can imagine that there are numerous interoperable implementations of the protocol, as with email. But in practice, most nodes run the reference implementation, and its developers are trusted by the community and have a lot of power.

2.2. DISTRIBUTED CONSENSUS

We've discussed, in a generic manner, centralization and decentralization. Let's now examine decentralization in Bitcoin at a more technical level. A key term that comes up throughout this discussion is *consensus*, specifically, *distributed consensus*. The key technical problem to solve in building a distributed e-cash system is achieving distributed consensus. Intuitively, you can think of our goal as decentralizing Scroogecoin, the hypothetical currency discussed in Chapter 1.

Distributed consensus has various applications, and it has been studied for decades in computer science. The traditional motivating application is reliability in distributed systems. Imagine you're in charge of the backend for a large social networking company, such as Facebook. Systems of this sort typically have thousands or even millions of servers, which together form a massive distributed database that records all actions that happen in the system. Each piece of information must be recorded on several different nodes in this backend, and the nodes must be in sync about the overall state of the system.

The implications of having a distributed consensus protocol reach far beyond this traditional application. If we had such a protocol, we could use it to build a massive, distributed key-value store that maps arbitrary keys, or names, to arbitrary values. A distributed key-value store, in turn, would enable many applications. For example, we could use it to build a distributed domain name system, which is simply a mapping between humanly intelligible domain names and IP addresses. We could build a public key directory, which is a mapping between email addresses (or some other form of real-world identity) and public keys.

That's the intuition of what distributed consensus is, but it is useful to provide a technical definition, as this will help us determine whether a given protocol meets the requirements.

Distributed consensus protocol. There are *n* nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:

- It must terminate with all honest nodes in agreement on the value.
- The value must have been generated by an honest node.

What does this mean in the context of Bitcoin? To understand how distributed consensus works in Bitcoin, remember that Bitcoin is a peer-to-peer system. When Alice wants to pay Bob, what she actually does is broadcast a transaction to all Bitcoin nodes that make up the peer-to-peer network (Figure 2.1).

Incidentally, you may have noticed that Alice broadcasts the transaction to all Bitcoin peer-to-peer nodes, but Bob's computer is nowhere in this picture. It's of course possible that Bob is running one of the nodes in the peer-to-peer network. In fact, if he wants to be notified that this transaction did in fact happen and that he has been paid, running a node might be a good idea. Nevertheless, there is no requirement that Bob be listening on the network; running a node is not necessary for Bob to receive the funds. The bitcoins will be his regardless of whether he's operating a node on the network.

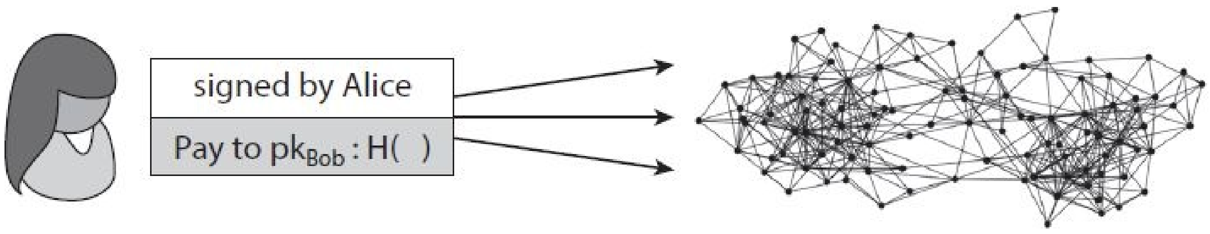

FIGURE 2.1. Broadcasting a transaction. To pay Bob, Alice broadcasts the transaction to the entire Bitcoin peer-to-peer network.

What exactly is it that the nodes might want to reach consensus on in the Bitcoin network? Given that a variety of users are broadcasting these transactions to the network, the nodes must agree on exactly which transactions were broadcast and the order in which these transactions occurred. This will result in a single, global ledger for the system. Recall that in Scroogecoin, for optimization, we put transactions into blocks (see Section 1.5). Similarly, in Bitcoin, consensus takes place on a block-by-block basis.

So at any given point, all nodes in the peer-to-peer network have a ledger consisting of a sequence of blocks, each containing a list of transactions that they have reached consensus on. Additionally, each node has a pool of outstanding transactions that it has heard about but that have not yet been included in the block chain. For these transactions, consensus has not yet happened, and so by definition, each node might have a slightly different version of the outstanding transaction pool. In practice, this occurs because the peer-to-peer network is not perfect, so some nodes may have heard about a transaction that other nodes have not yet heard about.

How exactly do nodes come to consensus on a block? One way to do this is as follows. At regular intervals (e.g., every 10 minutes), every node in the system proposes its own outstanding transaction pool to be included in the next block. Then the nodes execute some consensus protocol, where each node's input is its own proposed block. Now, some nodes may be malicious and put invalid transactions into their blocks, but we can assume that other nodes are honest. If the consensus protocol succeeds, a valid block will be selected as the output. Even if the selected block was proposed by only one node, it's a valid output as long as the block is valid. Now there may be some valid outstanding transaction that did not get included in the block, but this is not a problem. If some transaction somehow didn't make it into this particular block, it could just wait and get into the next block.

This approach bears some resemblence to how Bitcoin works, but it's not quite how it works. This approach has several technical problems. First, consensus in general is a hard problem, since nodes might crash or be outright malicious. Second, and specifically in the Bitcoin context, the network is highly imperfect. It's a peer-to-peer system, and not all pairs of nodes are connected to each other. There could be faults in the network because of poor Internet connectivity, for example, and thus running a consensus protocol in which all nodes must participate is not really possible. Finally, there's a lot of latency in the system, because it's distributed over the Internet.

Latency and Global Time

The Bitcoin protocol must reach consensus in the face of two types of

> obstacles: imperfections in the network (e.g., latency and nodes crashing) and deliberate attempts by some nodes to subvert the process.
>
> One particular consequence of this high latency is that there is no notion of global time. As a result, not all nodes can agree on a common ordering of events based simply on observing timestamps. So the consensus protocol cannot contain instructions of the form, “The node that sent the first message in step 1 must do *x* in step 2.” This simply will not work, because not all nodes will agree on which message was sent first in step 1 of the protocol.

Impossibility Results

The lack of global time heavily constrains the set of algorithms that can be used in the consensus protocols. In fact, because of these constraints, much of the literature on distributed consensus is somewhat pessimistic, and many impossibility results have been proven. One famous impossibility result concerns the *Byzantine Generals Problem*. In this classic problem, the Byzantine army is separated into divisions, each commanded by a general. The generals communicate by messenger to devise a joint plan of action. Some generals may be traitors and may intentionally try to subvert the process so that the loyal generals cannot arrive at a unified plan. The goal of this problem is for all loyal generals to arrive at the same plan without the traitorous generals being able to cause them to adopt a bad plan. It has been proven that this is impossible to achieve if one-third or more of the generals are traitors.

A much more subtle impossibility result, known by the names of the authors who first proved it, is the *Fischer-Lynch-Paterson* impossibility result. Under some conditions, which include the nodes acting in a deterministic manner, they proved that consensus is impossible with even a single faulty process.

Despite these impossibility results, there are some consensus protocols in the literature. One of the better known among these protocols is *Paxos*. Paxos makes certain compromises. On the one hand, it never produces an inconsistent result. On the other hand, it accepts the trade-off that under certain conditions, albeit rare ones, the protocol can fail to make any progress.

Breaking Traditional Assumptions

But there’s good news: these impossibility results were proven for a specific model. They were intended to study distributed databases, and this model doesn't carry over very well to the Bitcoin setting, because Bitcoin violates many of the assumptions built into the models. In a way, the results tell us more about the model than they do about the problem of distributed consensus.

Ironically, with the current state of research, consensus in Bitcoin works better in practice than in theory. That is, we observe consensus working but have not developed the theory to fully explain why it works. But developing such a theory is important, as it can help us predict unforeseen attacks and problems, and only when we have a strong theoretical understanding of how Bitcoin consensus works will we have strong guarantees of Bitcoin's security and stability.

What are the assumptions in traditional models for consensus that Bitcoin violates? First, it introduces the idea of incentives, which is novel for a distributed consensus protocol. This is only possible in Bitcoin because it is a currency and therefore has a natural mechanism to incentivize participants to act honestly. So Bitcoin doesn't quite solve the distributed consensus problem in a general sense, but it solves it in the specific context of a currency system.

Second, Bitcoin embraces the notion of randomness. As we shall see in the next two sections, Bitcoin's consensus algorithm relies heavily on randomization. Also, it does away with the notion of a specific starting point and ending point for consensus. Instead, consensus takes place over a long time, about an hour in the practical system. But even at the end of that time, nodes can't be certain that any particular transaction or a block has made it into the ledger. Instead, as time goes on, the probability increases that your view of any block will match the eventual consensus view, and the probability that the views will diverge goes down exponentially. These differences in the model are key to how Bitcoin gets around the traditional impossibility results for distributed consensus protocols.

2.3. CONSENSUS WITHOUT IDENTITY USING A BLOCK CHAIN

In this section we study the technical details of Bitcoin's consensus algorithm. Recall that Bitcoin nodes do not have persistent, long-term identities. This is another difference from traditional distributed consensus algorithms. One reason for this lack of persistent identities is that in a peer-to-peer system, there is no central authority to assign identities to participants and verify that they're not creating new nodes at will. The technical term for this is a *Sybil attack*. Sybils are just copies of nodes that a malicious adversary can create to make it look like there are a lot of different participants, when in fact all those pseudo-participants are really controlled by the same adversary. The other reason is that pseudonymity is inherently a goal of Bitcoin. Even if it were possible or easy to establish identities for all nodes or all participants, we wouldn't necessarily want to do that. Although Bitcoin doesn't give strong anonymity guarantees in that the different transactions that one makes can often be linked together, it does have the property that nobody is forced to reveal their real-life identity (e.g., their name or IP address) to participate. And that's an important property and a central feature of Bitcoin's design.

If nodes did have identities, the design would be easier. First, identities would allow us to put in the protocol instructions of the form, "Now the node with the lowest numerical ID should take some step." Without identities, the set of possible instructions is more constrained. But a second, much more serious, reason for nodes to have identities is for security. If nodes were identified and it weren't trivial to create new node identities, then we could make assumptions about the number of nodes that are malicious, and we could derive security properties based on those numbers. For both of these reasons, the lack of identities introduces difficulties for the consensus protocol in Bitcoin.

We can compensate for the lack of identities by making a weaker assumption. Suppose there is somehow an ability to pick a random node in the system. A good motivating analogy for this is a lottery or a raffle, or any number of real-life systems where it's hard to track people, give them identities, and verify those identities. What we do in those contexts is to give out tokens, tickets, or something similar. That enables us to later pick a random token ID and call on the owner of that ID. So for the moment, take a leap of faith and assume that it is possible to pick a random node from the Bitcoin network in this manner. Further assume, for the moment, that this algorithm for token generation and distribution is sufficiently smart so that if the adversary tries to create a lot of Sybil nodes, all those Sybils together will obtain only one token. Thus, the adversary is not able to multiply his power by creating new nodes. If you think this is a lot to assume, don't worry. In Section 2.4, we remove these assumptions and show in detail how properties equivalent to these are realized in Bitcoin.

Implicit Consensus

This assumption of random node selection makes possible something that we call *implicit consensus*. There are multiple rounds in our protocol, each corresponding to a different block in the block chain. In each round, a random node is somehow selected, and this node gets to propose the next block in the chain. There is no consensus algorithm for selecting the block, and no voting of any kind. The chosen node unilaterally proposes what the next block in the block chain will be. But what if that node is malicious? Well, a process exists for handling that, but it is an implicit one. Other nodes will implicitly accept or reject that block by choosing whether or not to build on top of it. If they accept that block, they will signal their acceptance by extending the block chain and including the accepted block. In contrast, if they reject that block, they will extend the chain by ignoring that block and building on the previous block that they accepted. Recall that each block contains a hash of the block that it extends. This is the technical mechanism that allows nodes to signal which block it is that they are extending.

Bitcoin consensus algorithm (simplified). This algorithm is simplified in that it assumes the ability to select a random node in a manner that is not vulnerable to Sybil attacks.

1. New transactions are broadcast to all nodes.
2. Each node collects new transactions into a block.
3. In each round, a *random* node gets to broadcast its block.
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures).
5. Nodes express their acceptance of the block by including its hash in the next block they create.

Let's now analyze why this consensus algorithm works. To do this, consider how a malicious adversary—call her Alice—may be able to subvert this process.

STEALING BITCOINS

Can Alice simply steal bitcoins belonging to another user at an address she doesn't control? No. Even if it is Alice's turn to propose the next block in the chain, she cannot steal other users' bitcoins. Doing so would require Alice to create a valid transaction that spends that coin.

This would require Alice to forge the owners' signatures, which she cannot do if a secure digital signature scheme is used. So as long as the underlying cryptography is solid, she's not able to simply steal bitcoins.

DENIAL-OF-SERVICE ATTACK

Let's consider another attack. Suppose that Alice really dislikes some other user Bob. Alice can then decide that she will not include any transactions originating from Bob's address in any block that she proposes to put in the block chain. In other words, she's denying service to Bob. Even though this is a valid attack that Alice can try to mount, luckily it's nothing more than a minor annoyance. If Bob's transaction doesn't make it into the next block that Alice proposes, he will just wait until an honest node has the chance to propose a block, and then his transaction will get into that block. So that's not really a good attack either.

DOUBLE-SPEND ATTACK

Alice may try to launch a double-spend attack. To understand how that works, let's assume that Alice is a customer of some online merchant or website run by Bob, who provides some online service in exchange for payment in bitcoins. Let's say Bob's service allows the download of some software. So here's how a double-spend attack might work. Alice adds an item to her shopping cart on Bob's website, and the server requests payment. Then Alice creates a Bitcoin transaction from her address to Bob's and broadcasts it to the network. Let's say that some honest node creates the next block, and includes this transaction in that block. So there is now a block that was created by an honest node that contains a transaction that represents a payment from Alice to the merchant Bob.

Recall that a transaction is a data structure that contains Alice's signature, an instruction to pay to Bob's public key, and a hash. This hash represents a pointer to a previous transaction output that Alice received and is now spending. That pointer must reference a transaction that was included in some previous block in the consensus chain.

Note, by the way, that there are two different types of hash pointers here that can easily be confused. Blocks include a hash pointer to the previous block that they're extending. Transactions include one or more hash pointers to previous transaction outputs that are being redeemed.

Let's return to how Alice can launch a double-spend attack. The latest block was generated by an honest node and includes a transaction in which Alice pays Bob for the software download. On seeing this transaction included in the block chain, Bob concludes that Alice has paid him and allows Alice to download the software. Suppose the next random node that is selected in the next round happens to be controlled by Alice. Since Alice gets to propose the next block, she could propose one that ignores the block that contains the payment to Bob and instead contains a pointer to the previous block. Furthermore, in the block that she proposes, Alice includes a transaction that transfers the very coins that she was sending to Bob to a different address that she herself controls. This is a classic double-spend pattern. Since the two transactions spend the same coins, only one of them can be included in the block chain. If Alice succeeds in including the payment to her own address in the block chain, then the transaction in which she pays Bob is useless, because it can never be included later in the block chain (Figure 2.2).

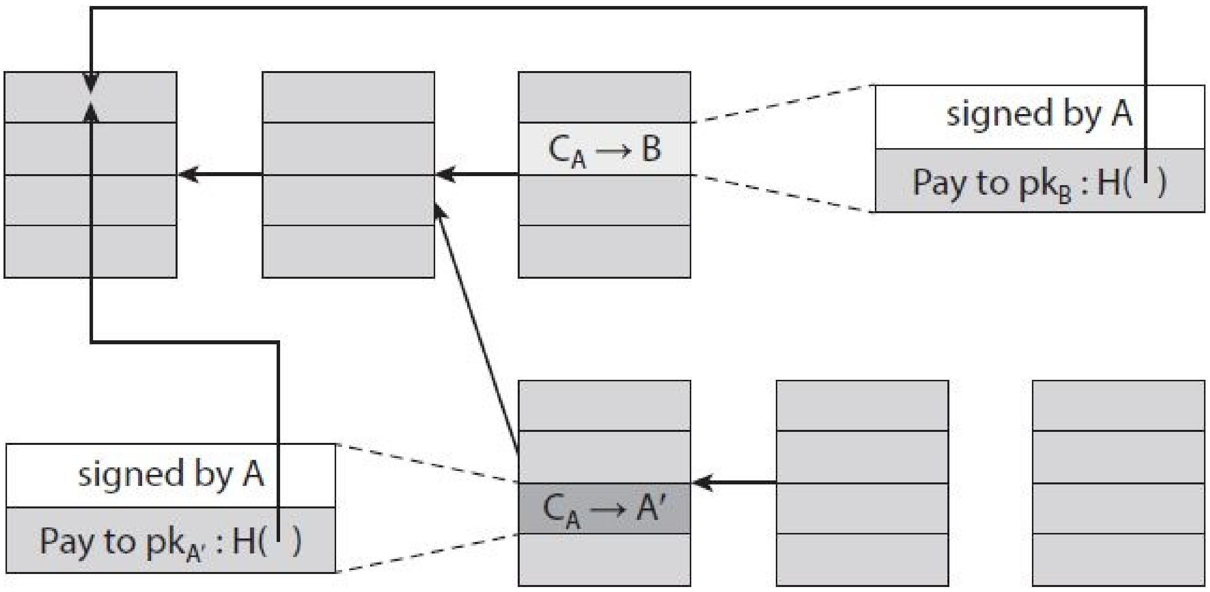

FIGURE 2.2. A double-spend attempt. Alice creates two transactions: one in which she sends Bob bitcoins, and a second in which she double spends those bitcoins by sending them to a different address, which she controls. As they spend the same bitcoins, only one of these transactions can be included in the block chain. The arrows between blocks are pointers from one block to the previous block that it extends by including a hash of that previous block within its own contents. C_A is used to denote a coin owned by Alice.

How do we know whether this double-spend attempt is going to succeed or not? Well, that depends on which block will ultimately end up on the long-term consensus chain—the one with the Alice → Bob transaction or the one with the Alice → Alice transaction. What determines which block will be included? Honest nodes follow the policy of extending the longest valid branch, so which branch will they extend? There is no right answer! At this point, the two branches are the same length—they only differ in the last block, and both of these blocks are valid. The node that chooses the next block then may decide to build on either one of them, and this choice will largely determine whether the double-spend attack succeeds.

A subtle point: from a moral point of view, there is a clear difference between the block containing the transaction that pays Bob and that containing the transaction in which Alice double spends those coins to her own address. But this distinction is only based on our knowledge of the story that Alice first paid Bob and then attempted to double spend. From a technological point of view, however, these two transactions are identical, and both blocks are equally valid. The nodes that are looking at this really have no way to tell which is the morally legitimate transaction.

In practice, nodes often follow a heuristic of extending the block that they first detected on the peer-to-peer network. But it's not a solid rule. And in any case, because of network latency, it could easily be that the block that a node first detected is actually the one that was created second. So there is at least some chance that the next node chosen to propose a block will extend the block containing the double spend. Alice could further try to increase the likelihood of this happening by bribing the next node to do so. If the next node does build on the double-spend block for whatever reason, then this chain will now be longer than the one that includes the transaction to Bob. At this point, the next honest node is much more likely to continue to build on this chain, since it is longer. This process will continue, and it will become increasingly likely that the block containing the double spend will be part of the long-term consensus chain. In contrast, the block containing the transaction to Bob is completely ignored by the network—it is now called a stale block or an *orphan block*.

Let's now reconsider this situation from Bob-the-merchant's point of view (Figure 2.3). Understanding how Bob can protect himself from this double-spending attack is a key part of understanding Bitcoin security. When Alice broadcasts the transaction that represents her payment to Bob, Bob is listening on the network and hears about this transaction even before the next block is created. If Bob were even more foolhardy than we previously described, he can complete the checkout process on the website and allow Alice to download the software right at that moment. That's called a *zero-confirmation transaction*. This leads to an even more basic double-spend attack than the one described before. Previously, for the double-spend attack to occur, we had to assume that a malicious actor controls the node that proposes the next block. But if Bob allows Alice to download the software before the transaction receives even a single confirmation on the block chain, then Alice can immediately broadcast a double-spend transaction, and an honest node may include it in the next block instead of the transaction that pays Bob.

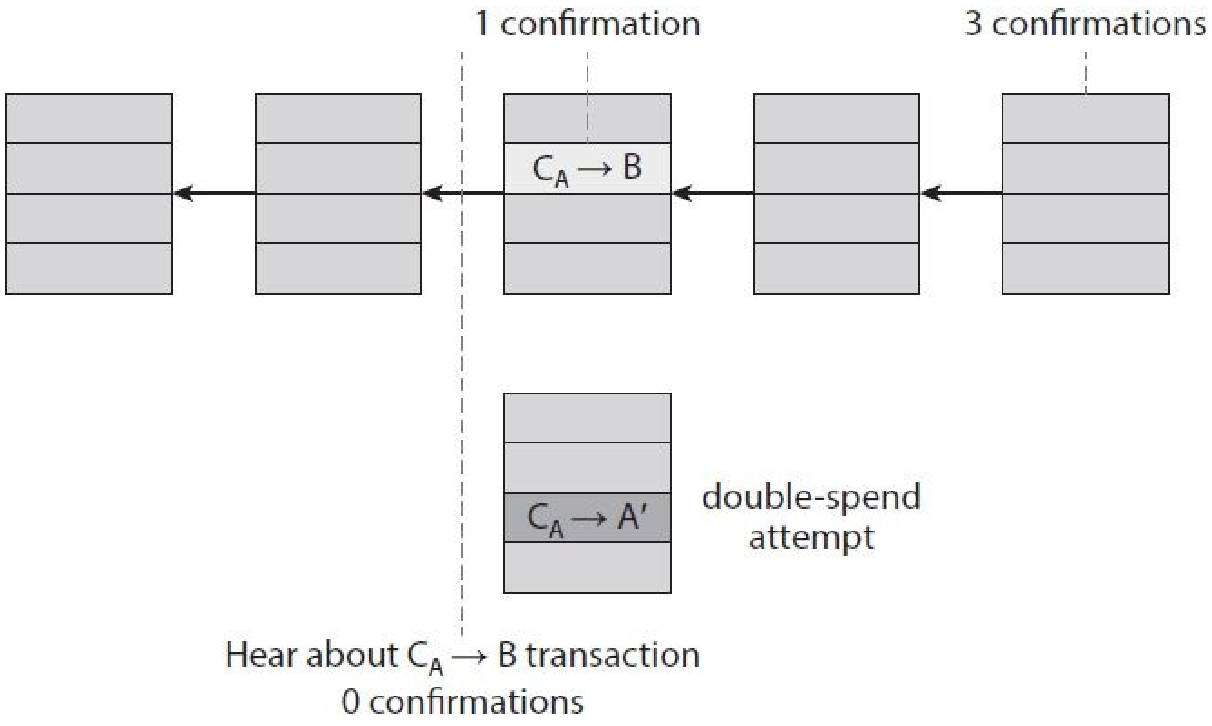

FIGURE 2.3. Bob the Merchant's point of view. This is what Alice's double-spend attempt looks like from Bob's viewpoint. To protect himself from this attack, Bob should wait to release the merchandise until the transaction with which Alice pays him is included in the block chain and has several confirmations.

However, a cautious merchant would not release the software to Alice even after the transaction was included in one block; he would continue to wait. If Bob sees that Alice successfully launches a double-spend attack, he realizes that the block containing Alice's payment to him has been orphaned. He should abandon the transaction and not let Alice download the software. Instead, if it happens that despite the double-spend attempt, the next several nodes build on the block with the Alice → Bob transaction, then Bob gains confidence that this transaction will be on the long-term consensus chain.

In general, the more confirmations a transaction gets, the higher the probability that it is going to end up on the long-term consensus chain. Recall that honest nodes always extend the longest valid branch that they find. The chance that the shorter branch with the double spend will catch up to the longer branch becomes increasingly tiny as the latter grows longer than any other branch. This is especially true if only a minority of the nodes are malicious—for a shorter branch to catch up, several malicious nodes would have to be picked in close succession.

In fact, the double-spend probability decreases exponentially with the number of confirmations. So, if the transaction that you're interested in has received k confirmations, then the probability that a double-spend transaction will end up on the long-term consensus chain goes down exponentially as a function of k. The most common heuristic that's used in the Bitcoin ecosystem is to wait for six confirmations. There is nothing really special about the number six. It's just a good trade-off between the amount of time you have to wait and your guarantee that the transaction you're interested in ends up on the consensus block chain.

To recap, protection against invalid transactions is entirely cryptographic. But it is enforced by consensus, which means that if a node does attempt to include a cryptographically invalid transaction, then the only reason that transaction won't end up in the long-term consensus chain is because a majority of the nodes are honest and won't include an invalid transaction in the block chain. In contrast, protection against double spending is purely by consensus. Cryptography has nothing to say about this, and two transactions that represent a double-spend attempt are both valid from a cryptographic perspective. But it's the consensus that determines which one will end up on the long-term consensus chain. And finally, you're never 100 percent sure that a transaction you're interested in is on the consensus branch. But this exponential probability guarantee is rather good. After about six transactions, there's virtually no chance that you're going to be deceived.

2.4. INCENTIVES AND PROOF OF WORK

In the previous section, we took a basic look at Bitcoin's consensus algorithm and developed a good intuition for why we believe that it's secure. But recall from the beginning of the chapter that Bitcoin's decentralization is partly a technical mechanism and partly clever incentive engineering. So far we've mostly looked at the technical mechanism. Now let's talk about the incentive engineering built into Bitcoin.

We asked you to take a leap of faith earlier in assuming that we're able to pick a random node and, perhaps more problematically, that at least 50 percent of the time, this process will pick an honest node. This assumption of honesty is particularly problematic if there are financial incentives for participants to subvert the process, in which case we can't really assume that a node will be honest. The question then becomes: Can we give nodes an incentive for behaving honestly?

Consider again the double-spend attempt after one confirmation (see Figure 2.2). Can we somehow penalize the node that created the block with the double-spend transaction? Well, not really. As mentioned earlier, it's hard to know which is the morally legitimate transaction. But even if we did, it's still hard to punish nodes, since they don't have identities. So instead, let's flip the question around and ask: Can we reward each of the nodes that created the blocks that did end up on the long-term consensus chain? Well, again, since those nodes don't reveal their real-world identities, we can't quite mail them cash to their home addresses. If only there were some sort of digital currency that we could use instead … you can probably see where this is going. We're going to use bitcoins to incentivize the nodes that created these blocks.

Let's pause for a moment. Everything described so far is just an abstract algorithm for achieving distributed consensus and is not specific to the application. Now we're going to use the fact that the application we're building through this distributed consensus process is in fact a currency. Specifically, we're going to incentivize nodes to behave honestly by paying them in units of this currency.

Block Reward

How is this done? Two separate incentive mechanisms are used in Bitcoin. The first is the *block reward*. According to the rules of Bitcoin, the node that creates a block gets to include a special transaction in that block. This transaction is a coin-creation transaction, analogous to CreateCoins in Scroogecoin, and the node can also choose the recipient address of this transaction. Of course that node will typically choose an address belonging to itself. You can think of this as a payment to the node in exchange for the service of creating a block on the consensus chain.

As of 2015, the value of the block reward is fixed at 25 bitcoins. But it actually halves with every 210,000 blocks created. Based on the rate of block creation, the rate halves roughly every four years. We're now in the second period. For the first four years of Bitcoin's existence, the block reward was 50 bitcoins; now it's 25. And it's going to keep halving. This has some interesting consequences, which we address below.

You may be wondering why the block reward incentivizes honest behavior. It may appear, based on what we've said so far, that this node gets the block reward regardless of whether it proposes a valid block or behaves maliciously. But this is not true! Think about it—how will this node collect its reward? That will only happen if the block in question ends up on the long-term consensus branch, because just like every other transaction, the coin-creation transaction will only be accepted by other nodes if it ends up on the consensus chain. That's the key idea behind this incentive mechanism. It's a subtle but powerful trick. It incentivizes nodes to behave in whatever way they believe will get other nodes to extend their blocks. So if most of the network is following the longest-valid-branch rule, it incentivizes all nodes to continue to follow that rule. That's Bitcoin's first incentive mechanism.

We mentioned that every 210,000 blocks (or approximately four years), the block reward is cut in half. In Figure 2.4, the slope of this curve is going to keep halving. This is a geometric series, and you might know that it means that there is a finite sum of bitcoins created by this mechanism. It works out to a total of 21 million bitcoins.

Note that this is the only way in which new bitcoins can be created. There is no other coin-generation mechanism, which is why 21 million is a final and total number (as the rules stand now, at least) for how many bitcoins there can ever be. This block reward will run out in 2140, as things stand now. Does that mean that the system will stop working in 2140 and become insecure, because nodes no longer have the incentive to behave honestly? Not quite. The block reward is only the first of two incentive mechanisms in Bitcoin.

Transaction Fees

The second incentive mechanism is the *transaction fee*. The creator of any transaction can choose to make the total value of the transaction outputs less than the total value of its inputs. Whoever creates the block that first puts that transaction into the block chain gets to collect the difference, which acts a transaction fee. So if you're a node that is creating a block containing, say, 200 transactions, then the sum of those 200 transaction fees is paid to the address that you put into that block. The transaction fee is purely voluntary, but we expect, based on our understanding of the system, that as the block reward starts to run out, it will become more and more important, almost mandatory, for users to include transaction fees to maintain a reasonable quality of service. To a certain degree, this is already starting to happen now. But it is currently unclear precisely how the system will evolve; it really depends on a lot of game theory, which hasn't been fully worked out yet. This is an interesting area of open research in Bitcoin.

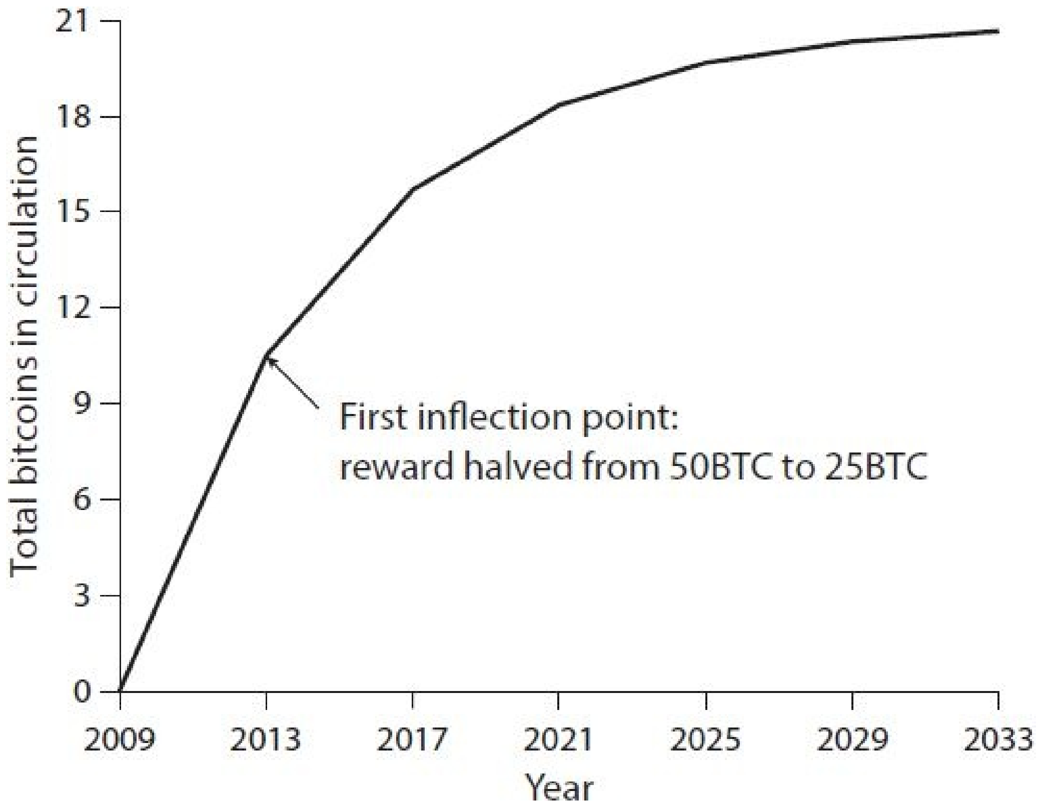

FIGURE 2.4. Total supply of bitcoins with time. The block reward is cut in half every 4 years, limiting the total supply of bitcoins to 21 million. This is a simplified model and the actual curve looks slightly different, but it has the same 21 million limit.

A few problems still remain with the consensus mechanism as described here. The first major one is the leap of faith that we asked you to take that somehow we can pick a random node. Second, we've created a new problem by giving nodes these incentives for participation. The system can become unstable as the incentives cause a free-for-all, where everybody wants to run a Bitcoin node in the hope of capturing some of these rewards. And a third one is an even trickier version of this problem: an adversary might create a large number of Sybil nodes to try and subvert the consensus process.

Mining and Proof of Work

All these problems are related, and all have the same solution, which is called *proof of work*. The key idea behind proof of work is that we approximate the selection of a random node by instead selecting nodes in proportion to a resource that we hope that nobody can monopolize. If, for example, that resource is computing power, then it's a proof-of-work system. Alternately, it could be in proportion to ownership of the currency, which is known as *proof of stake*. Although it's not used in Bitcoin, proof of stake is a legitimate alternate model that is used in other cryptocurrencies. We'll see more about proof of stake and other proof-of-work variants in Chapter 8.

But back to proof of work. Let's clarify what it means to select nodes in proportion to their computing power. This can be thought of as allowing nodes to compete with one another by using their computing power, which will result in nodes automatically being picked in proportion to that capacity. Yet another view of proof of work is that we're making it moderately hard to create new identities. It's a sort of tax on identity creation and therefore on the Sybil attack. This might all appear a bit vague, so let's look at the details of the proof-of-work system used in Bitcoin, which should clarify the concept.

Bitcoin achieves proof of work using *hash puzzles*. To create a block, the node that proposes that block is required to find a number (a nonce; see Section 1.1), such that when you concatenate the nonce, the previous hash, and the list of transactions that make up the block and then take the hash of this whole string, then that hash output should be a number that falls in a target space that is quite small in relation to the much larger output space of that hash function. We can define such a target space as any value falling below a certain target value. In this case, the nonce will have to satisfy the following inequality:

$$H(nonce \parallel prev\_hash \parallel tx \parallel tx \parallel \ldots \parallel tx) < target$$

As we have seen, normally a block contains a series of transactions that a node is proposing. In addition, a block also contains a hash pointer to the previous block. (We are using the term “hash pointer” loosely. The pointer is just a string in this context, as it need not tell us where to find this block. We can find the block by asking other peers on the network for it. The important part is the hash that both acts as an ID when requesting other peers for the block and lets us validate the block once we have obtained it.) In addition, we’re now requiring that a block also contain a nonce. The idea is that we want to make it moderately difficult to find a nonce that satisfies this required property, which is that hashing the whole block together, including that nonce, is going to result in a particular type of output. If the hash function satisfies the puzzle-friendliness property from Chapter 1, then the only way to succeed in solving this hash puzzle is to just try enough nonces one by one until you get lucky. So specifically, if this target space were just 1 percent of the overall output space, you would have to try about 100 nonces before you are likely to get lucky. In reality, the size of this target space is not nearly as high as 1 percent of the output space. It’s much, much smaller than that, as we will see shortly.

This notion of hash puzzles and proof of work completely does away with the requirement to magically pick a random node. Instead, nodes are simply independently competing to solve these hash puzzles all the time. Once in a while, one of them will find a random nonce that satisfies this property. That lucky node then gets to propose the next block. By this means, the system is completely decentralized. Nobody is deciding which node gets to propose the next block.

Difficult to Compute

There are three important properties of hash puzzles. The first is that they need to be quite difficult to compute. We said moderately difficult, but you’ll see why this actually varies with time. As of 2015, the difficulty level is over 10^{20} hashes per block. In other words, the size of the target space is less than $1/10^{20}$ of the size of the output space of the hash function. Searching the output space thus involves a lot of computation—it's out of the realm of possibility for a commodity laptop, for example. Because of this, only some nodes even bother to compete in this block creation process. This process of repeatedly trying and solving these hash puzzles is known as *Bitcoin mining,* and the participating nodes are called *miners*. Even though technically anybody can be a miner, power has become concentrated in the mining ecosystem due to the high cost of mining.

Parameterizable Cost

The second property we want is that the cost should be parameterizable rather than fixed for all time. This is accomplished by having all the nodes in the Bitcoin peer-to-peer network automatically recalculate the target (i.e., the size of the target space as a fraction of the output space) every 2,016 blocks. They recalculate the target in such a way that the average time between successive blocks produced in the Bitcoin network is about 10 minutes. With a 10-minute average time between blocks, 2,016 blocks works out to two weeks. In other words, the recalculation of the target happens roughly every two weeks.

Consider what this means. Suppose you are a miner, and you've invested a certain fixed amount of hardware into Bitcoin mining. But the overall mining ecosystem is growing, more miners are coming in, or they're deploying faster and faster hardware, which means that over a two-week period, slightly more blocks are going to be found than expected. So nodes will automatically readjust the target, and the amount of work that you have to do to find a block will increase. So if you invest a fixed amount in hardware, the rate at which you find blocks actually depends on what other miners are doing. A very nice formula captures this: the probability that any given miner, Alice, is going to win the next block is equivalent to the fraction of global hash power that she controls. So if Alice has mining hardware that's about 0.1 percent of total hash power, she will find roughly one in every 1,000 blocks.

What is the purpose of this readjustment? Why do we want to maintain this 10-minute invariant? The reason is quite simple. If blocks were to come very close together, then there would be a lot of inefficiency, and we would lose the optimization benefits of being able to put many transactions in a single block. There is nothing magical about the number 10, and if you changed from 10 minutes to 5 minutes, the system would probably work just fine. There's been a lot of discussion about the ideal block latency that *altcoins* (alternative cryptocurrencies) should have. But despite some disagreements about the ideal latency, everybody agrees that it should be a fixed amount. It cannot be allowed to go down without limit. That's why Bitcoin features automatic target recalculation.

Two Models of Miner Behavior

In the research fields of distributed systems and computer security, it is common to assume that some percentage of nodes are honest and to show that the system works as intended even if the other nodes behave arbitrarily. That's basically the approach we've taken here, except that we weight nodes by hash power when computing the majority. The original Bitcoin white paper contains this type of analysis as well.

But the field of game theory provides an entirely different—and arguably more sophisticated and realistic—way to determine how a system will behave. In this view, we don't split nodes into honest and malicious. Instead, we assume that *every* node acts according to its incentives. Each node picks a (randomized) strategy to maximize its payoff, taking into account other nodes' potential strategies. If the protocol and incentives are designed well, then most nodes will follow the rules most of the time. "Honest" behavior is then just one strategy among many, and we attach no particular moral salience to it.

In the game-theoretic view, the big question is whether the default miner behavior is a *Nash equilibrium*, that is, whether it represents a stable situation in which no miner can realize a higher payoff by deviating from honest behavior. This question is still contentious and is an active area of research.

The way that this cost function and proof of work is set up allows us to reformulate our security assumption. Here's where we finally depart from the last leap of faith that we asked you to take earlier. Instead of assuming that somehow the majority of nodes are honest in a context where nodes don't even have identities and not being clear about what "honesty" means, we can now state crisply that many attacks on Bitcoin are infeasible if the majority of miners, weighted by hash power, are following the protocol—that is, are honest. This is true because if most miners, weighted by hash power, are honest, then competition for proposing the next block will automatically ensure at least a 50 percent chance that the next block to be proposed at any point is coming from an honest node.

Solving hash puzzles is probabilistic, because nobody can predict which nonce is going to solve the hash puzzle. The only way to solve the puzzle is to try nonces one by one and hope that one succeeds. Mathematically, this process is called a *Bernoulli trial*. A Bernoulli trial is an experiment with two possible outcomes, and the probability of each outcome occurring is fixed between successive trials. Here, the two outcomes are (1) the hash falls in the target, and (2) it does not. Assuming that the hash function behaves like a random function, the probability of those two outcomes is fixed. Typically, nodes try so many nonces that Bernoulli trials, a discrete probability process, can be well approximated by a continuous probability process known as a *Poisson process*, one in which events occur independently at a constant average rate. The end result is that the probability density function showing the relative likelihood of the time until the next block is found looks like the graph in Figure 2.5.

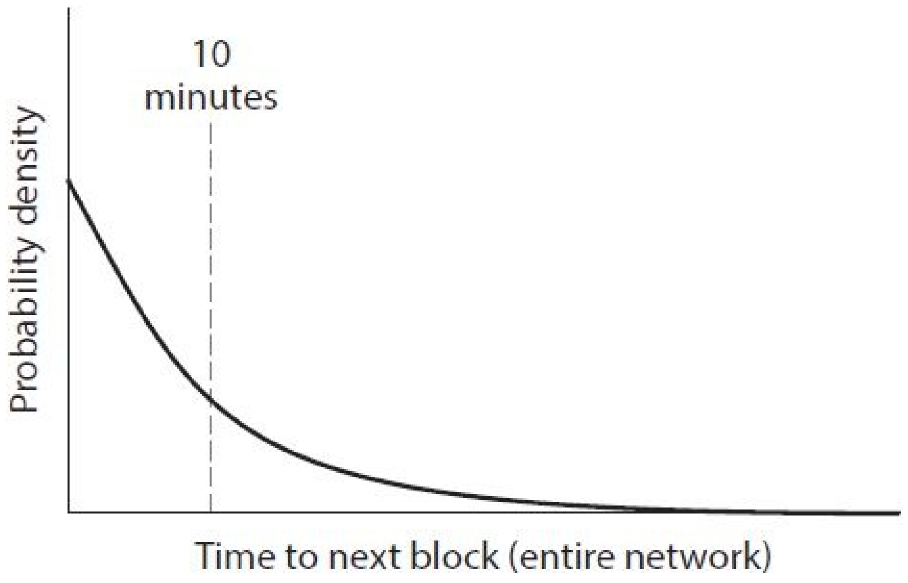

FIGURE 2.5. Probability density function of the time until the next block is found.

This is known as an *exponential distribution*. Some small probability exists that if a block has been found now, the next block is going to be found very soon, say, within a few seconds or a minute. And there is also some small probability that it will take a long time, say, an hour, to find the next block. But overall, the network automatically adjusts the difficulty so that the inter-block time is maintained at an average, long term, of 10 minutes. Notice that Figure 2.5 shows how frequently blocks are going to be created by the entire network, regardless of which miner actually finds the block.

If you're a miner, you're probably interested in how long it will take you to find a block. What does this probability density function look like? It will have the same shape but a different scale on the *x*-axis. Again, it can be represented by a nice equation.

For a specific miner:

$$\text{mean time to next block} = \frac{\text{10 minutes}}{\text{fraction of hash power}}$$

If you have 0.1 percent of the total network hash power, this equation states that you're going to find blocks once every 10,000 minutes, which is just about a week. Not only is your mean time between blocks going to be high, but the variance of the time between blocks found by you is also going to be high. This has some important consequences that are discussed in Chapter 5.

Trivial to Verify

Now we turn to the third important property of this proof-of-work function: it is trivial to verify that a node has computed proof of work correctly. Even if it takes a node, on average, 10^{20} tries to find a nonce that makes the block hash fall below the target, that nonce must be published as part of the block. It is thus trivial for any other node to look at the block contents, hash them all together, and verify that the output is less than the target. This is quite an important property, because, once again, it allows us to get rid of centralization. We don't need any centralized authority verifying that miners are doing their job correctly. Any node or any miner can instantly verify that a block found by another miner satisfies this proof-of-work property.

2.5. PUTTING IT ALL TOGETHER

Cost of Mining

Let's now look at mining economics. We mentioned that it is quite expensive to operate as a miner. At the current difficulty level, finding a single block takes computing about 10^{20} hashes, and the block reward is about 25 bitcoins, which is a sizable amount of money at the current exchange rate. These numbers allow for an easy calculation of whether it's profitable for one to mine, and we can capture this decision with a simple statement:

```
If
        mining reward > mining cost
then the miner makes a profit
where
        mining reward = block reward + tx fees
        mining cost = hardware cost + operating costs (electricity,
        cooling, etc.)
```

Fundamentally, the miner obtains her mining rewards from block rewards and transaction fees. The miner asks herself how these rewards compare to the total expenditure, which is the hardware and electricity cost.

But there are some complications to this simple equation. The first is that, as you may have noticed, the hardware cost is a fixed cost, whereas the electricity is a variable cost that is incurred over time. Another complication is that the reward obtained by miners depends on the rate at which they find blocks, which depends on not only the power of their hardware, but also on the ratio of their hash rate to the total global hash rate. A third complication is that the costs that the miner incurs are typically denominated in dollars or some other traditional currency, but their reward is denominated in bitcoins. So this equation has a hidden dependence on Bitcoin's exchange rate at any given time. And finally, so far we've assumed that the miner is interested in honestly following the protocol. But the miner might choose to use some other mining strategy instead of always attempting to extend the longest valid branch. So this equation doesn't capture all the nuances of the different strategies that the miner can employ. Actually analyzing whether it makes sense to mine is a complicated game theory problem that's not easily answered.

There Is No Such Thing as One Bitcoin

Bitcoin doesn't have fixed denominations like U.S. dollar bills, and in particular, there is no special designation of "1 bitcoin." Bitcoins are just transaction outputs, and in the current rules, they can have an arbitrary value to eight decimal places of precision. The smallest possible value is 0.00000001 BTC (bitcoins), which is called 1 *satoshi*.

At this point, we have a pretty good picture of how Bitcoin achieves decentralization. We now recap the major points and put it all together for an even better understanding.

Let's start with identities. As we've learned, real-world identities are not required to participate in the Bitcoin protocol. Any user can create any number of pseudonymous key pairs at any moment. When Alice wants to pay Bob in bitcoins, the Bitcoin protocol does not specify how Alice learns Bob's address. Given these pseudonymous key pairs as identities, transactions are basically messages broadcast to the Bitcoin peer-to-peer network that are instructions to transfer coins from one address to another. Bitcoins are just transaction outputs, and we will discuss this in much more detail in Chapter 3.

The goal of the Bitcoin peer-to-peer network is to propagate all new transactions and new blocks to all Bitcoin peer nodes. But the network is highly imperfect and does a best-effort attempt to relay this information. The security of the system doesn't come from the perfection of the peer-to-peer network. Instead, the security comes from the block chain and the consensus protocol that we devoted much of this chapter to studying.

When we say that a transaction is included in the block chain, what we really mean is that the transaction has achieved numerous confirmations. No fixed number of confirmations is necessary before we are sufficiently convinced of the transaction's inclusion, but six is a commonly used heuristic. The more confirmations a transaction has received, the more certain you can be that this transaction is part of the consensus chain. Orphan blocks (blocks that don't make it into the consensus chain) often arise. Various reasons can lead to a block being orphaned. The block may contain an invalid transaction, or a double-spend attempt. Orphaning can also just be a result of network latency. That is, two miners may simply end up finding new blocks within just a few seconds of each other. So both of these blocks were broadcast nearly simultaneously on the network, and one of them will inevitably be orphaned.

We next looked at hash puzzles and mining. Miners are special types of nodes that decide to compete in this game of creating new blocks. They're rewarded for their effort in terms of both newly minted bitcoins (the block reward) and existing bitcoins (transaction fees), provided that other miners build on their blocks. A subtle but crucial point: say that Alice and Bob are two different miners, and Alice has 100 times as much computing power as Bob. This does not mean that Alice will always win the race against Bob to find the next block. Instead, Alice and Bob have a probability ratio of finding the next block of 100 to 1. In the long term, Bob will find, on average, 1 percent of the number of blocks that Alice finds.

We expect that miners will typically be somewhere close to the economic equilibrium in the sense that the expenditure they incur in terms of hardware and electricity will be roughly equal to the rewards they obtain. The reason is that if a miner is consistently making a loss, she will probably stop mining. In contrast, if mining is very profitable given typical hardware and electricity costs, then more mining hardware would enter the network. The increased hash rate would lead to an increase in the difficulty, and each miner's expected reward would drop.

This notion of distributed consensus permeates Bitcoin. In a traditional (fiat) currency, consensus does come into play to a limited extent. Specifically, a consensus process determines the exchange rate of the currency. That is certainly true in Bitcoin as well. We need consensus about the value of bitcoins. But in Bitcoin, additionally, we need consensus on the state of the ledger, which is what the block chain accomplishes. In other words, even the accounting of how many bitcoins you own is subject to consensus. When we say that Alice owns a certain amount or number of bitcoins, what we actually mean is that the Bitcoin peer-to-peer network, as recorded in the block chain, considers the sum total of all Alice's addresses to own that number of bitcoins. That is the ultimate nature of truth in Bitcoin: ownership of bitcoins is nothing more than other nodes agreeing that a given party owns those bitcoins.

Finally, we need consensus about the rules of the system, because occasionally these rules have to change. Two types of changes are made to the rules of Bitcoin, known respectively as *soft forks* and *hard forks*. We defer a detailed discussion of the differences to Chapters 3 and 7.

Getting a Cryptocurrency off the Ground

Another subtle concept is that of *bootstrapping*. A tricky interplay takes place among three different ideas in Bitcoin: the security of the block chain, the health of the mining ecosystem, and the value of the currency. We obviously want the block chain to be secure for Bitcoin to be a viable currency. For the block chain to be secure, an adversary must not be able to overwhelm the consensus process. This in turn means that an adversary cannot create a lot of mining nodes and assume 50 percent or more of the new block creation.

But when will these conditions be met? A prerequisite is having a healthy mining ecosystem made up of largely honest, protocol-following nodes. But what's a prerequisite for that—when can we be sure that a lot of miners will put a lot of computing power into participating in this hash-puzzle-solving competition? They're only going to make the effort if the exchange rate of bitcoins is pretty high, because the rewards miners receive are denominated in bitcoins, whereas their expenditures are in dollars. So the higher the value of the currency, the more incentivized these miners are going to be.

But what ensures a high and stable value of the currency? That can only happen if users in general trust the security of the block chain. If they believe that the network could be overwhelmed at any moment by an attacker, then Bitcoin will not have much value as a currency. So you have an interlocking interdependence among the security of the block chain, a healthy mining ecosystem, and the exchange rate.

Because of the cyclical nature of this three-way dependence, the existence of each of these is predicated on the existence of the others. When Bitcoin was first created, none of these three conditions was met. There were no miners other than Nakamoto himself running the mining software (see the Foreword). Bitcoin didn't have a lot of value as a currency. And the block chain was, in fact, insecure, because not much mining was going on, and anybody could have easily overwhelmed this process.

There's no simple explanation for how Bitcoin went from not having any of these properties to having all three of them. Media attention was part of the story—the more people hear about Bitcoin, the more they become interested in mining. And the more they get interested in mining, the more confidence people will have in the security of the block chain, because then more mining activity is going on, and so forth. Incidentally, every new altcoin that wants to succeed also has to somehow solve this problem of pulling itself up by its bootstraps.

The 51 Percent Attack

Finally, let's consider what would happen if consensus failed and there was in fact a *51 percent attacker* (one who controls a majority of the mining power in the Bitcoin network). We'll consider a variety of possible attacks and see which ones can actually be carried out by such an attacker.

First of all, can this attacker steal coins from an existing address? As you may have guessed, the answer is no, because stealing from an existing address is not possible unless you subvert the cryptography. It's not enough to subvert the consensus process. This is not completely obvious. Let's say the 51 percent attacker creates an invalid block that contains an invalid transaction that tries to steal bitcoins from an existing address that the attacker doesn't control and transfer them to his own address. The attacker can pretend that it's a valid transaction and keep building on this block. He may even succeed in making this block part of the longest branch. But the other, honest nodes are simply not going to accept this block with an invalid transaction and are going to keep mining based on the last valid block that they found in the network. So a fork in the chain will occur.

Now imagine this from the point of view of the attacker, who is trying to spend these invalid coins and sends them to some merchant Bob as payment for goods or services. Bob is presumably running a Bitcoin node himself, and it will be an honest node. Bob's node will reject that branch as invalid, because it contains an invalid transaction. It has been determined to be invalid, because the signatures don't check out. So Bob's node will simply ignore the longest branch, because it's an invalid branch. And because of that, subverting consensus is not enough. You have to subvert cryptography to steal bitcoins. So we conclude that this attack is not possible for a 51 percent attacker.

Note that this is only a thought experiment. If there were, in fact, actual signs of a 51 percent attack, what would probably happen is that the developers would notice it and react. They would update the Bitcoin software, and we might expect that the rules of the system, including the peer-to-peer network, might change to make it more difficult for this attack to succeed. But we can't quite predict that. So we're working in a simplified model, where a 51 percent attack happens, but no changes or tweaks are made to the rules of the system.

Let's consider another attack. Can the 51 percent attacker suppress some transactions? Let's say there is some user, Carol, whom the attacker really doesn't like. The attacker knows some of Carol's addresses and wants to make sure that no coins belonging to any of those addresses can be spent. Is that possible? Since he controls the consensus process of the block chain, the attacker can simply refuse to create any new blocks that contain transactions from one of Carol's addresses. The attacker can further refuse to build on blocks that contain such transactions. However, he can't prevent these transactions from being broadcast to the peer-to-peer network, because the network doesn't depend on the block chain or on consensus, and we're assuming that the attacker doesn't fully control the network. The attacker cannot stop the transactions from reaching the majority of nodes, so even if the attack succeeds, it will at least be apparent that the attack is happening.

Can the attacker change the block reward? That is, can the attacker start pretending that the block reward is, instead of 25 bitcoins, say, 100 bitcoins? This is a change to the rules of the system, and because the attacker doesn't control the copies of the Bitcoin software that all honest nodes are running, this is also not possible. The reason is similar to that explaining why the attacker cannot include invalid transactions. Other nodes will simply not recognize the increase in the block reward, and the attacker will thus be unable to spend them.

Finally, can the attacker somehow destroy confidence in Bitcoin? Well, let's imagine what would happen. If there were a variety of double-spend attempts, situations in which nodes did not extend the longest valid branch, and other attempted attacks, then people likely would decide that Bitcoin is no longer acting as a decentralized ledger that they can trust. They would lose confidence in the currency, and we might expect that the exchange rate of Bitcoin would plummet. In fact, if it were known that a party controls 51 percent of the hash power, then it's possible that people would lose confidence in Bitcoin even if the attacker is not necessarily trying to launch any attacks. So it is not only possible, but in fact likely, that a 51 percent attacker of any sort will destroy confidence in the currency. Indeed, this is the main practical threat if a 51 percent attack were ever to materialize. Considering the amount of expenditure that the adversary would have to put into attacking Bitcoin and achieving a 51 percent majority, none of the other attacks that we described really make sense from a financial point of view.

Hopefully, at this point you understand how decentralization is achieved in Bitcoin. You should have a good command of how identities work in Bitcoin, how transactions are propagated and validated, the role of the peer-to-peer network in Bitcoin, how the block chain is used to achieve consensus, and how hash puzzles and mining work. These concepts provide a solid foundation and a good launching point for understanding a lot of the more subtle details and nuances of Bitcoin, which we're going to see in subsequent chapters.

FURTHER READING

The Bitcoin white paper is:

Nakamoto, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System." 2008. Available at https://bitcoin.org/bitcoin.pdf.

The original application of proof of work is:

Back, Adam. "Hashcash—A Denial of Service Counter-measure." 2002. Available at http://www.hashcash.org/papers/hashcash.pdf.

The Paxos algorithm for consensus is:

Lamport, Leslie. "Paxos Made Simple." *ACM Sigact News* 32(4), 2001: 18–25.