Understanding Blockchains: Not Too High-Level, Not Too Detailed

By E. Andrew Boyd

16 December 2022

Introduction

What is a blockchain, really? Typical descriptions either gloss over the definition or quickly become mired in detail. Here I attempt a description that's somewhere in the middle; a description I found myself comfortable with. Hopefully, it's at a level that some others have been looking for. Note: an understanding requires some familiarity with computer science.

A Blockchain

As much as the term "ledger" gets thrown about, a blockchain really is the equivalent of a ledger. Think of those old black books accountants kept on their shelves, filled with pages for recording transactions, transactions like "we sent $100 to company A" or "we received $50 from client B."

In the parlance of blockchains, pages of the ledger are referred to as *blocks*. They are fixed amounts of computer storage (Bitcoin seeks to achieve 1 Mb per block) where transactions are recorded. One block corresponds to one page of a ledger, and once a block is filled it is archived: no information in that block can be modified. The magic of blockchain technology is that even though information in a blockchain can be accessible to anyone with a computer, it can't be changed once archived.

A block may contain many different transactions. Blockchains can contain really any information we care to record, but for simplicity we'll stick with the model most people think of -- transactions of things being bought and sold with a cryptocurrency like bitcoin. "Person A spent one bitcoin to purchase an equivalent amount of another cryptocurrency called ether." "Person B spent one bitcoin to purchase U.S. dollars."

A fundamental characteristic is that a blockchain ledger doesn't come with a fixed number of pages. Instead, new pages are added to the back of a ledger over time. Presently, the most common process of adding new pages to a ledger (blocks to the end of a blockchain) is called *mining*.

The term mining is a bit of a misnomer. Finding blocks that can meet the requirements to be added to a blockchain is computationally difficult and requires considerable computational power. So miners – people who use computers to search for new blocks – are rewarded with cryptocurrency for their efforts. Miners don't mine cryptocurrency itself; they mine blocks to add to the end of a blockchain. New cryptocurrency enters the system as payment for finding a new block, not because miners found new cryptocurrency.

Where is a Blockchain?

Unlike a physical ledger, a blockchain doesn't reside in one specific location. Instead, a complete copy of a blockchain resides on many different computers with connections to the internet. This brings us to another term that is widely used when describing blockchain technology: *distributed*. Blockchains are distributed ledgers. Computers that contain a copy of a blockchain are known as nodes of that blockchain.

Any computer with enough power and memory can become a node on a blockchain if its owner simply downloads free software and installs it. Many middle-of-the-road PCs are completely capable of serving as a node on one or more blockchains. Reasons for becoming a node run from having a direct connection to a blockchain – you don't have to request information from another computer – to simple bragging rights.

The Big Question

How do we ensure that no one modifies a blockchain? That no one goes into a blockchain and changes the information contained in it? Returning to our ledger metaphor, how do we know someone can't use an eraser and pencil to change entries in a ledger?

Bear in mind that there is no single copy of a blockchain. So changing information in "a blockchain" requires tricking all of the nodes containing a copy of a blockchain into believing that a modified blockchain is actually the original. On the surface this doesn't seem like such a difficult problem to avoid, especially if there are hundreds or thousands or tens of thousands of nodes in a blockchain. One bad actor tricking all the others seems a monumental task.

However, the incentive to make changes exists for those actors who desire to steal cryptocurrency: simply create transactions that transfer cryptocurrency from someone else's account into their own. Given that every actor participating in a blockchain has an incentive to falsify records, ensuring the integrity of a blockchain isn't as simple as it might appear.

There are potentially many ways to address the integrity problem, and more than one way has been proposed. Here we are considering the most prominent method currently in use.

Ensuring the Information in a Blockchain isn't Corrupted: Digital Signatures

Blockchains operate by writing information only to the last block. Once that block can no longer hold additional information, a new block is added to the end of a blockchain and the block that was just filled is archived. It can be read by anyone, but if anyone tries to change the information in the archived block it sets off a red flag.

This red flag comes in the form of a digital fingerprint, more commonly known as a *hash value*. A hash function is a function that takes a large amount of data and reduces it to a unique, scrambled code. This code is the hash value. Such cryptographic codes have been used throughout history and remain essential in, among other things, transmitting e-commerce transactions across the internet.

Let's look at hash values and the functions that generate them before turning to how they're used to secure the data in a blockchain.

One common function, known as SHA-1, was designed by the U.S. National Security Agency. Suppose we wanted to use SHA-1 to create the hash value of the following string of letters.

The quick brown fox jumps over the lazy dog

Providing this string of letters to the SHA-1 hash function, the output would be

2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

There are a few things to note about the output. First, it has exactly 40 characters. Second, it is made up of only the numbers 0-9 and the letters a-f (this is related to the hexadecimal representation of binary numbers but is irrelevant for our discussion). This is always the case for the SHA-1 hash function. Suppose we were to add a period to this string of characters and calculate the new hash value.

The quick brown fox jumps over the lazy dog.

In this case, the output is

408d94384216f890ff7a0c3528e8bed1e0b01621

Observe that with just a very small change in the input – adding a period to the end of the character string – the resulting hash value looks nothing like the first value. The SHA-1 function is designed to have this property, and it helps demonstrate a property that's part of what we want in a cryptographic hash function: given the output, it's very, very difficult to reconstruct the input. In fact, a good cryptographic hash function should be designed so that you can't go backward. If you have an output from a hash function but you don't know what the input to the function was, you must try random values on the input side until you find one that matches the output. The probability of doing so is so small that it really can't be done.

As might be expected, the process behind creating good hash functions requires some mathematical knowledge. But it's possible to create good hash functions that are both simple to implement and for which hash values can be calculated very quickly.

Ensuring the Information in a Blockchain isn't Corrupted: Using Hash Functions

Consider the case where the last block in a blockchain, say, block 116, is full and we're ready to archive the information and start a new block, block 117. As part of this process we'll use the information in block 116 as input to our hash function. We then place the resultant hash value in block 117. Note that this is done every time a block is filled and archived, so every block from 1 to 116 has calculated its hash value and placed that value in the next higher block.

Suppose an intruder now changes the information in any block in the chain, say, block 63. If we were to compute the hash value for block 63 and compare it to the hash value stored in block 64, it wouldn't match. Anyone else who then sought to look at the information in block 63 would check the hash value stored on block 64 and would know the data on block 63 could no longer be trusted.

Of course, the obvious question is why the intruder doesn't just go down the line, computing new hash values for blocks 64, 65, 66 and so on so that they're all in accord. In fact, without the introduction of one additional idea the intruder could easily do just that.

The additional idea that keeps this from happening is, in my estimation, rather ingenious, though it’s difficult to tease from much of the material on the web.

Putting Things Together

We have so far assumed that we take the data in a block, compute its hash value, and place that value in the next block in the chain. However, that’s not exactly correct. We’re going to restrict acceptable hash values to values that start with a fixed number of zeroes, say, 15. So the hash value

000000000000000e18fda057782390abb7d7619a3

would be acceptable, while the hash values

6e80754da12b14e0cae84839d6707850412fcc76

and

0000008832ba1820338ee01b9d46b9edd8bc1f4d

would not. We define a *valid* hash value as one that starts with at least 15 zeroes.

Before getting to the important question of *why* we might want to do this, it’s worth considering the even more basic question of *how* we might do this. Cryptographic hash functions take an input and create a hash value, and we know very little about what the hash value will look like.

The way to ensure that the hash value is valid is to tweak the input a little bit, run it through the hash function, and look at the hash value. If it has the desired number of leading zeroes you can use it as a valid hash value. If not, you just keep trying.

Several issues must be raised at this point. First, what does it mean to tweak the input a little bit? Isn’t the data in a block immutable? Not entirely. Each block has areas where the data can be changed without altering any important information in the block. Think of writing comments in a special comments section. Additionally, many blockchains include a *nonce*. Derived from “number used once,” a nonce is an area specially set aside for the sole purpose of tweaking the input a little.

The second issue is how someone goes about finding one of these rare hash values. Won’t that take a lot of time and computational power? The answer is yes but with a caveat. No single computer is searching for a good way to adjust the input of the block seeking an acceptable hash value. Many computers are working on the problem simultaneously.

This is where the miners come in. Miners around world are constantly seeking a new block to fill with whatever transactions are ready to be archived. When a block with tweaked data is found with a valid hash value, the block data, complete with the comments and nonce that made its hash value valid, are added to a blockchain.

Now that we've answered the question of how we can turn to the question of why. The answer is that it makes it impossible for someone to tamper with a blockchain. Recall the scenario before the valid hash values were introduced. An intruder could change the information in a block, write a new hash value (valid or invalid) in the next block – thus changing the hash value in that block. Changing all the hash values down the chain the intruder would have covered their work.

Now, however, the scenario is different. The intruder can't use just any hash value. Hash values are required to be valid. Since the leading zeroes requirement is public knowledge, if someone were to look in a block and find a hash value that didn't meet the leading zeroes requirement, it would be a sign that somewhere in the blocks preceding it the blockchain had been corrupted. It had been corrupted because the chain was initially put together with blocks whose hash values met the leading zeroes requirement.

With this observation, we see that we now have a blockchain that can keep transactions safe in a distributed environment, without a single individual overseeing the transactions.

What has been presented thus far provides a good foundation digging more deeply into blockchain technology. However, I'd like to make a few additional comments without trying to cover every aspect blockchains..

Mempools

Before transactions become part of a blockchain they're maintained in what is known as a *mempool* – a holding place in computer memory. Each node maintains its own mempool, and while the transactions they hold are similar from one mempool to another, they're not guaranteed to be identical.

Transactions aren't kept sequentially in the order they arrive at a node, but receive different treatment depending on how much the person behind the transaction is willing to pay in fees. The higher the fee, the more the transaction is prioritized. As demand for transactions increases, fees tend to rise for different levels of service.

While transactions wait in a mempool, nodes that are also miners begin seeking a new block to add to the blockchain. As new transactions arrive, they simply use the full transaction list as the input to the hashing function. When a valid hash value is discovered, the miner that discovers it includes it at the end of a blockchain together with the transaction data that formed the basis of the hash value, and then sends this information to all the nodes associated with the blockchain. Nodes verify that the block is indeed valid, update their copy of a blockchain, and remove any transactions in this new block from their mempools. There are certainly many more details about mempools I'm not covering and haven't looked into. These include, among other things, maintaining the integrity of transactions while they're in a mempool, the degree to which mempools coordinate, specifics about the fee structures, and how the transfer of the fees is handled.

One of the interesting situations that can occur is that a relatively empty block can be added to a blockchain. As soon as a block is added to a blockchain, miners immediately begin searching for a new valid block with whatever remains in their mempools. If the mempool is small, and a miner is lucky enough to create a new block very shortly after the last block was discovered, a relatively empty block is added to a blockchain. If the mempool is empty, it's even possible for an empty block to be added to a blockchain. Miners don't get paid for adding transactions to a blockchain, they get paid for adding blocks to a blockchain.

Probability

Blockchains operate under the assumption that one actor cannot modify a blockchain because they would need to modify not only one block but create enough new blocks to replace all subsequent blocks in the chain. This is considered impossible from a practical standpoint because finding blocks with valid hash values is time consuming and there are many miners competing to find new blocks. A bad actor simply wouldn't be able to create enough new blocks in time without corruption of the chain first being uncovered.

However, this is true only if mining capacity is distributed among many competing miners. If, for example, one miner or a colluding group of miners were generating, say, 90 percent of new blocks, they could create any necessary blocks with high probability and in turn manipulate and corrupt the chain. It is difficult to assess the probability of such an event, but it certainly isn't "vanishingly small."

Managing the Speed of Block Discovery

Another creative design aspect of blockchains is the way the speed of block discovery is controlled. Bitcoin, for example, seeks to have miners create one new block every ten minutes. Given the many different miners and no direct means of knowing the power of their mining equipment, the problem might at first seem challenging. The actual solution, however, is remarkably simple.

Records are kept to determine the average length of time between block discoveries. Suppose new blocks are being discovered over a long period of time at an average speed of one new block every ten minutes. However, looking just at more recent discoveries, it's discovered that the average is trending lower to a rate of one block every nine minutes. Statistically, miners are mining faster than before, perhaps because new miners have entered the world of mining. How does Bitcoin increase the average to ten minutes again?

It does so by increasing the number of leading zeroes required on valid hash values. The longer the string of zeroes, the harder it is to find a valid hash value, and thus the longer the process will take. Reducing the length of time between discoveries is achieved by reducing the number of leading zeroes.

Energy Consumption

Mining cryptocurrency requires computers solving one simple problem: guess an input by modifying the data in a block that's ready to be archived, calculate a hash value for that block, and see if it has enough leading zeroes. The only purpose for expending this energy is to maintain the integrity of a blockchain.

While I've seen many different estimates, the energy expended on crypto mining amounts to anywhere from one-half to one percent of world energy consumption. That's a lot of energy, especially when one considers the other things it could be used for.

Reducing Energy Consumption

The key to making the distributed ledger nature of blockchains work is a means to make certain that data in a blockchain remains uncorrupted. The method described here, with miners seeking new blocks, is known as *proof-of-work*. Is it possible that some creative individual could come up with an alternative that's more energy efficient?

Yes. An alternative that is actively being explored is referred to as *proof-of-stake*. A description of proof-of-stake is a topic unto itself and one I will not cover here, though I found the following video a reasonably good high-level description that doesn't entirely gloss over details (though it's still too high-level for my taste). It also has some history on proof-of-work.

https://www.youtube.com/watch?v=M3EFi_POhps

Are there other alternatives to proof-of-work and proof-of-stake? Quite possibly.