Overview of Blockchains and Cryptocurrencies

Justin Wyss-Gallifent

April 17, 2022

0.1 Introduction

This is a medium-level outline of how blockchains and bitcoin work.

0.2 Hash Functions

A *hash function* is effectively a function which takes an arbitrary nonnegative input number and creates a nonnegative output number less than some predetermined value. That output number is officially called a *message digest* but the term *hash* is often used.

A hash function has the property that given the input number it's easy to calculate the output number but given a target output number it's very very hard to know what the input number should be in order to obtain that output number.

It's like baking a cake in the sense that given some ingredients it's easy to make the cake but given the cake it's hard to know exactly what the ingredients should be in order to get that cake.

0.3 Blockchains

A *blockchain* is a sequence of blocks - a *block* is just a collection of information:

$$B_0, B_1, B_2, ..., B_n$$

Each block essentially contains:

- A timestamp of when it was created.
- Some data.
- A hash of the entire previous block. The zeroth block, called the *genesis block*, does not contain this, because there is no previous block.

Note that these connections makes it difficult to change the data in a block because changing the data in a block would involve changing the hash in the next block, which would then require changing the hash in the block after that block, and so on, all the way to the end of the blockchain.

While it's possible to store a blockchain on just one computer, usually the exact same blockchain is stored over multiple (thousands or more) computers, called *nodes*, all of which are communicating with one another as a network, and all of which are running the same software which manages this blockchain.

Because the blockchain is stored on multiple nodes it is impossible to change the blockchain data on the copy of the blockchain on just one node because the other nodes would see that it's out of sync and ignore it. To change the blockchain data would essentially require changing it on 51% of the nodes so that they became the majority.

Instead, the blockchain can have blocks added to it by the software that each runs, according to a protocol mentioned shortly.

0.4 Storing Value

Suppose the data in each block contains transaction data. For example suppose there are just three blocks and they say:

B_0:

- 1000 ₿ created for Address 1.

B_1:

- Send 800 ₿ from Address 1 to Address 2.
- 700 ₿ created for Address 3.

B_2:

- Send 100 ₿ from Address 2 to Address 1.
- Send 100 ₿ from Address 2 to Address 3.

Then we know:

- Address 1 contains $1000 - 800 + 100 = 300$ ₿,
- Address 2 contains $800 - 100 - 100 = 600$ ₿.
- Address 3 contains $700 + 100 = 800$ ₿.

0.5 Updating the Blockchain

0.5.1 How?

Now then, since the blockchain is distributed over lots of nodes, how do new transactions get added?

Well, suppose I am Address 1 and I wish to send 5 ₿ to Address 2, I create a message containing this information:

Hey! Address 1 gives 5 ₿ to Address 2.

I send this message to one (or more) nodes on the network which then each sends it on to 3-4 more nodes, which then do the same, so soon my request propagates to every single node. Each node verifies that Address 1 actually owns that many bitcoin and if the transaction is valid then it adds that transaction to a collection of transactions which it is packaging up into a new *prospective block*. Notice that all nodes are doing this so they all have the same collection of new transactions which they are packaging into the same new prospective block.

Note: This is not exactly true as I'll elaborate on later, but believing it's true is sufficient to understand what's happening.

Once this new prospective block reaches 1MB in size (this is hardcoded), one of those nodes gets the privilege of officially adding it to the blockchain on every node. But who? And why are they doing all this in the first place?

0.5.2 Why?

The node which gets to add the new block to the blockchain earns bitcoin. Currently (April 2022) that's 6.25 ₿. This gets halved every 4 years (this is hardcoded and will happen next in 2024). The node also earns transaction fees (more bitcoin) from the transactions themselves - those fees are paid by the senders.

0.5.3 Who?

Each node takes the collection of transactions as well as a current timestamp, and then tries to find a number, called a *nonce* (in this case a 32-bit number), such that when all of these are combined together, converted to a number and hashed, the resulting hash is less than some difficulty target that the entire network is aware of (managed by the software). Doing this is hard in the sense that rolling a billion-sided die and getting a number less than 10 is hard. Meaning it's unlikely, and each node has to try lots of nonces until it hopefully gets the right one. So the nodes are all choosing random nonces (think - rolling dice), hashing everything together, and hoping to be the first one to hit the target hash.

Note that lots of sources say that the nodes are "solving complicated math problems" but this is complete rubbish. They are not solving anything in any intelligent sense. They are generating random nonces, hashing, and crossing their silicon fingers.

When a node gets the nonce right and gets below the target hash it sends the entire new block around to all the other nodes in the network. This block contains the nonce, the timestamp, the transactions and also contains "Address ME just earned the current block creation reward and also the transaction fees for the included transactions!". The other nodes can see it was right (it includes the nonce) and see that it was first (it includes the timestamp) and so they stop their calculation (they lost!), add this new block to their copies of the blockchain, and start working on the next prospective block.

0.6 Some Technical Details

0.6.1 Variation in Prospective Blocks

Each node doesn't necessarily receive the new transactions in exactly the same order since multiple transactions are being spread out through the network at any given moment. Therefore each block may in fact be building a slightly different prospective block.

In addition when a prospective block is in the process of being validated there are still new transactions arriving and so nodes may start building even newer prospective blocks chained to their older prospective blocks.

What happens when one node finally validates its prospective block is that all the other nodes throw away their prospective blocks and any transactions which have not been included in the actual validated prospective block are put back into the network for inclusion into a future prospective block.

This is why a bitcoin transaction can take more than the 10 minutes it takes, on average, to mine a new block. That transaction may have got into a node's prospective block which didn't get validated during that iteration and it had to wait for a later prospective block.

0.6.2 Checking Bitcoin Totals

If Address 1 wants to send 100 ₿ somewhere a node doesn't actually need to check all the way back in the blockchain to count how many bitcoins Address 1 owns. Instead it just looks back far enough to ensure that at least 100 ₿ has been deposited to Address 1 but not spent.

It's like me checking whether you have at least $100 in your bank account. I don't need to know how much you have, if I know that $30 was deposited yesterday and $85 was deposited this morning and if you haven't spent any, then you certainly have enough.

0.6.3 The Difficulty Target

Every 2016 blocks (this is hardcoded) the difficulty target is changed by all the nodes in the network. The goal is to ensure, as much as possible, that a new block is created about every 10 minutes. So for example as mining hardware improves, the difficulty target should be harder to obtain for any given node. Likewise is half the nodes went offline then the difficulty target should be easier to obtain for any given node.

0.6.4 Securing the Transactions

Next, how are the transactions themselves secured? For example if I am Address 1 what stops me from broadcasting this message to the network:

Hey! Address 3 gives 5000000000 Ƀ to Address 1.

You have a private key and a public key. Your bitcoin address is a much-hashed version of your public key. See details of this later. When you send a message to the network you:

1. Create the transaction m which sends money to someone else (that's all you can do).
2. Sign it with your private key: $D_{YOU}(m)$
3. Send it to the network along with your public key.

Each node then:

1. Propagates this information to several other nodes, which will then do the same.
2. Un-signs the message using your public key: $E_{YOU}\left(D_{YOU}(m)\right) = m$
3. Looks at how much money you want to send and who to send it to.
4. Checks to see that you have that much money to spend.
5. Assuming all is well, adds that transaction to its prospective block.

0.6.5 The Hash Algorithms

Several hashes come into play:

- For hashing each block and incorporating it into the next block, SHA-256 is used.
- For hashing a block along with the nonce for validation purposes, SHA-256 is used.
- For generation of a bitcoin address, the public/private key pair are a ECDSA pair (see below), the public key is hashed via SHA-256, then hashed by RIPEMD-160, then prepended with a version prefix byte, then hash via SHA-256 again, then hash via SHA-256 yet again, take the first four bytes from this result and append it to the RIPEMD-160 plus version prefix byte, as a checksum.

0.6.6 Cryptographic Stuff

The securing keys are elliptic curve keys generated using the Secp256k1 elliptic curve. This is the curve $y^2 = x^3 + 7$ taken over the finite field F_p where p is the prime:

$$\begin{aligned} p &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \\ &= 115792089237316195423570985008687907853269984665640564039457584007908834671663 \end{aligned}$$

This elliptic curve is particularly nice in that it allows for efficient computation.

The key structure is based upon the elliptic curve discrete logarithm problem which is essentially the elliptic curve version of the ElGamal signature scheme.

0.7 Proof of Work v Proof of Stake

The method given above for being the node selected to submit the block is called *proof of work*, the idea being that that node has to do work and prove it (find the nonce) in order to win. This is highly energy-intensive.

An alternate approach to this taken by some cryptocurrencies (Cardano and Ethereum 2.0) is known as *proof of stake*. Let's focus on Ethereum. Note that Ethereum is the network and Ether is the currency.

In the proof of stake approach, nodes who wish to validates blocks are required to "stake" some quantity of ETH. This essentially means they are required to have some investment in the process.

A single node is then chosen as the block miner using a transparent pseudo-random approch from all of the nodes who have put in stakes. The actual selection weights each node according to the amount of ETH staked and how long they have been staked and then throws in a random element in the sense that the two criteria generate a target (in the same sense as Bitcoin) but the target is different for each node. A node with more stake will have a higher (easier to hit) target.

A number of other nodes are then chosen as validators. The block miner then shares the node with the validators and if two-thirds of them validate the block it it added to the blockchain.

In this case not only the miner but also the validators earn an ETH reward.

0.8 Solidity

Solidity is specific to the Ethereum blockchain but it puts forth a good final point. The data stored in each block is just data. In the Ethereum blocks it also can include computer code written in Solidity as well as data beyond transaction data. When a block is validated the code is run and this code can then perform certain functions, updating data as well as generating more transactions.

Some examples:

- The Solidity code can generate new coins which live inside the Ethereum blockchain. These are typically called *tokens*. The Basic Attention Token is one of these.
- The Solidity code can monitor external sources such as the stock market and can then execute code based upon that.
- The Solidity code can send a certain amount of ETH to somone else but only at a certain date and time.
- Etc.