6.
Mining - Part I - The Principles

To fully understand what mining is, and why it's important we first need to cover some of the basic principles and functions that are used in mining. We will cover these principles in Part I. Part II will then tie these principles together and explain mining as a whole in relation to our example. This particular example is focussed primarily around Bitcoin mining. Almost all cryptocurrencies work fundamentally in the same way, but some have slight differences that we will go over once the concept of mining is a little more clear.

Proof Of Work:
You may have heard of the term 'Proof Of Work'. The easiest way to visualize the Proof Of Work protocol is essentially a math puzzle. A computational puzzle which requires a significant amount of computational work to be completed for the puzzle to be solved, yet it has to be quickly verifiable. In other words, the outcome of the Proof Of Work, the Proof, is essentially a number, or a hash, which we can quickly be verified to be either correct or incorrect, but would need a significant amount of computation to solve.
This is probably the most difficult and least intuitive part of the protocol. But it may be a little bit easier to understand by visualizing it with an example. Imagine this puzzle as a shuffled Rubik's cube. It takes work and time to solve it, but you can very quickly verify that it is correct by simply observing it and checking that all 6 sides of the cube have their appropriate matching colours. This verification can be done without any real work on the verification side aside from checking that the colours match. After verifying that the cube is solved you can be sure that the person handing you the cube has put in time and effort to come up with the correct solution to solve it. This is very similar to the Proof Of Work puzzle, it is extremely difficult and time consuming for a computer to solve, but very quick and easy for every node in the network to verify.

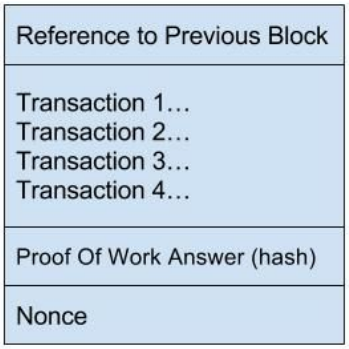
Proof Of Work is the concept that allows a new block (set of transactions) to be added to the distributed ledger (the blockchain). But before we explain this we need to go over a few more technical principles that you need to be familiar with, first of which being: what is a 'block' and what information does it hold?

As the players in our distributed network are making transactions, these transactions will be broadcasted out into the network. 'Miners' listen and wait for these transactions to occur, they then validate the transactions and bundle them together into a 'block'. So a block is really just a bundle of transactions.
The amount of transactions that every one block can hold is determined by the rules of the blockchain that is being used.The Bitcoin blockchain allows a block size of 1 megabyte for each

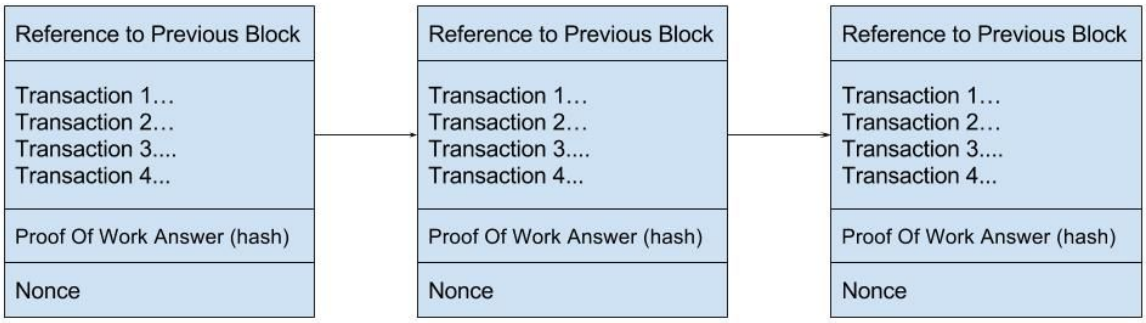
block, however, other cryptocurrencies may have different block sizes varying the amount of transactions that can be held in each block.

Now there are a few more data items in a block than the items listed here (such as the block size, block header, transaction counter, etc.), but as the principle is the same, and we are trying to keep our example easy to follow, in our example each block contains:

- A bundle of transactions that have been verified (signed via digital signature, no overspending, etc.).
- A reference to the previous block. As a blockchain is an ever growing and expanding chain (or list), the new block contains a reference to the block that comes immediately before it.
- The answer to the Proof Of Work puzzle. This is the answer that needs to be correct before the block is accepted, and appended to the blockchain. This is our solved rubik's cube.
- The Nonce. A Nonce defined as an arbitrary number that is only used on one occasion, or used only once (**N** - number, **ONCE** - used once). And this is the interesting part, this is the number that the computer will use to try and solve the Proof Of Work puzzle. We will explain this shortly, but for it to make sense first we need to explain one more technical item.



Using the Proof Of Work protocol then allows a miner to append a new block (a list of new transactions) to be added to the existing chain of blocks (the ledger). This is why instead of a distributed ledger we refer to this as the blockchain, even though they are essentially one and the same.



The first thing the miner will do after bundling the transactions and forming the block is creating a hash for that block. We will cover what a hash is more detail momentarily, but it works very similar to the digital signatures we described earlier.
In the previous chapter when signing or creating the digital signature we learned that any kind of change to the transaction or the message that is being encrypted will completely change the resulting signature code. The same principle applies here. When a hash is created for the block, which is outcome of hashing the block into a fixed number via a hashing function, just like the Sign() function in the last chapter. Currently most of the data items in the block are static. Once the transactions have been verified, they will not change, the reference to the previous block will not change, and neither will any of the other information held within the block, with the exception of the Nonce.
The **Nonce** gives us a number field that we can alter in any way we want, simply to adjust the outcome of an encryption algorithm. This is the only purpose for the nonce, it only serves to alter the resulting hash of a hashing function.

What is a hash function, and what is a hash?
A hash function, or hash algorithm is a function that takes an input, which can be anything, a string of characters, a file, or in our case a block of transactions and turns it into a fixed length hexadecimal number. Although it may seem similar, hashing functions should not be confused with encryption algorithms. Encryption algorithms allow you to encrypt a set of data, and then decrypt it with a key, allowing you to get your original data back. Hashing algorithms don't work this way, hashing algorithms are essentially a one-way street, or in other words, if you create a hash (the number output of a hashing function) by passing a file through a hashing function, it is impossible to recreate that file from the hash. It can be thought of as a signature for a given set of text or a data file. This hash number has a fixed length, which means that regardless of how big the input file is, the length of the output will never change.
Hashes have a lot of different purposes, but one of the key characteristics is that it allows us to prove data integrity quickly by providing us with a relatively short string of characters that are unique to the input data we gave the hash function.
For example, if we put the characters 'abc' through a SHA 256 hash function, we will receive a hash, or string of characters that is 64 characters long. This string of 64 characters is a 256 bit

number in hexadecimal format (hexadecimal is a numerical system which uses a base 16 system, or 16 characters).
If we put a 400 page book through that same hashing function, we will also receive a 64 character hash. The important thing to understand about hashes is that this fixed length hash will completely change as soon as the input changes even slightly. So in our example of hashing a 400 page book, if we were to change a single letter in that book, the hash will completely change. Comparing hashes therefore allows us to very quickly determine if data has been changed or tampered with.
There are many different types of hashing algorithms. Specifically, we are referring to the SHA256 hashing algorithm. SHA256 is the name given to the hashing function, SHA is an acronym for Secure Hash Algorithm, the 256 means that the function will create an outcome of a fixed size of 256 bits. In other words, regardless of how long or short the input for the function is, a paragraph of text, or simply the number '5', the outcome will always be a 256 bit number (represented in hexadecimal this is number is 64 characters long).
This can be a difficult concept to understand so let's go through some examples.

Using the SHA256 hashing function:

Hashing the value 'abc' will have an outcome of:

edeaaff3f1774ad2888673770c6d64097e391bc362d7d6fb34982ddf0efd18cb

Every time we hash 'abc' the hash will look identical, allowing us by comparison to determine that the input string was the string 'abc'.

If we were to make a small change to this input string and change it to 'abd', the outcome will be:

a52d159f262b2c6ddb724a61840befc36eb30c88877a4030b65cbe86298449c9

You can see that changing just one character in the input has changed the output dramatically.

Similarly, hashing the entire content of this book gives us the output:

87018a7eb50d42a93424db5080653c464e15c0448cb447166c33159c90a215c9

As you can see, it doesn't matter how large or small the input data is, the outcome will always be a fixed size of 256 bits, being 64 characters in hexadecimal. Changing one letter of one word of this book, will give us the outcome:

05d074ec0893e416cd734bfdd927ba94c3115309a9b5a5b2c29ece7ca7ca06d1

By comparing these two hash values you can immediately determine that they are different, and therefore that the input data of the two was different. This is a lot easier than taking two seemingly identical books and checking if one contains a word that is spelled slightly different.

Now if we change the word in the book back to what it was originally, and run it back through our hash function, we again get the original hash, allowing us to easily determine that the book is identical to what it was before.

87018a7eb50d42a93424db5080653c464e15c0448cb447166c33159c90a215c9

Now that we have a basic understanding of the fundamental principles behind hashing, we can return to the question of what the Proof Of Work protocol is. Just a quick recap, let's recall the contents of a block:

- Transactions
- Reference to previous block
- Answer to the Proof Of Work puzzle
- The Nonce

As we mentioned, the transactions contained within the block and the reference to the previous block, will not change, they are static data. As we just learned if we put static data through a hash function without any change, the hash will always result in the same output hash. However, changing the input data ever so slightly will generate a completely different, seemingly random hash.
This is the sole purpose of the Nonce. It is simply a number that we can adjust to change the outcome of our hash function to receive a desired hash.

So why is this important? Manipulating the hash of a block is the Proof Of Work mining puzzle.

The mining puzzle that needs to be solved is that every block that is to be mined needs to have a SHA 256 hash value that has a certain amount of leading zeros. Going back to our earlier analogy of a rubiks cube, a random hash value such as:

a52d159f262b2c6ddb724a61840befc36eb30c88877a4030b65cbe86298449c9

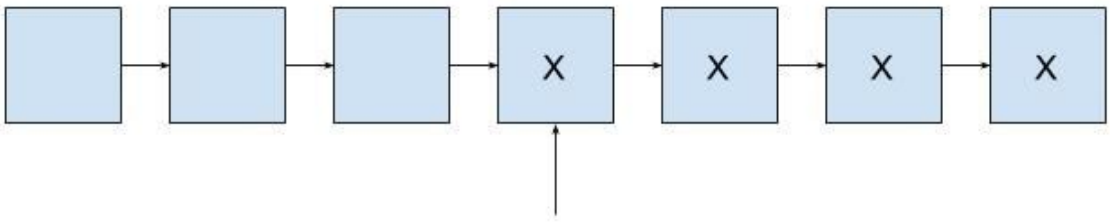
Would be the equivalent of a scrambled cube, but you can immediately determine if the hash puzzle is solved if the hash contains the required amount of leading zeros, this would be the solved rubik's cube.

0000000000000000db724a61840befc36eb30c88877a4030b65cbe86298449c9

Because we don't know the inner workings of how the SHA256 algorithm creates its hashes, and because the contents of each new block will always be different, the only approach we have to solve this puzzle is to pick random numbers as the Nonce, run the block contents including the nonce through the hash function, and check if it fulfills the leading 0s requirement. This is random checking on the miners part. There is no way for one miner to have any kind of advantage over other miners other than luck, and obviously to some degree, computing power.

This is the basic principle behind mining, it's the random checking of numbers to find a hash value that fulfills this required criteria. As simple as this process may sound it serves multiple purposes.

First, it provides us with data integrity of the blockchain itself. As we know, the slightest change in the data input will completely change the outcome hash. We mentioned that a new block that is to be mined contains the reference to its previous block, or latest block in the chain. This reference is actually the hash value of that given block.
So if you think about what this means, the entire blockchain is essentially a hash of a hash, of a hash, etc. as the hashing of the newest block always contains the new transactions, and the hash of the previous block. So if someone were to tamper with a any block in the blockchain, regardless of how far back in the chain, that block, along with every block that came after it, would suddenly evaluate to a different hash value that would no longer fulfill the leading 0s requirement, indicating that it was tampered with. This also means we could also immediately be able to determine which block was the first to change.



Block is attacked, transaction is changed. Thereby changing its hash value, and every following block's hash value.

Second, finding the correct number (Nonce) to create a hash with a given number of leading 0s is extremely difficult and work intensive for a computer. This in turn makes it impossible for a hacker to 'hack' the blockchain, as changing a single transaction would require them to re-solve every following block's hashing puzzle, and broadcast the blocks out before the rest of the network has detected the intrusion. This is impossible. We will go into this in a later chapter but at this point we should also mention that the underlying Bitcoin blockchain has to this point

never been hacked. You have probably heard reports of ‘Bitcoin having been hacked’, but these hacks were of the endpoints of Bitcoin, Bitcoin exchanges, wallets, or other devices used with Bitcoin. The blockchain has never been hacked.