Playing With Blocks: The Basics of Blockchain Databases (Part 2 - Blockchain for Techies)

Author	josh
Date	2018-11-04 23:24:44

Overview

In our non-technical overview of blockchain, we discussed what a blockchain database is - a distributed, cryptographically secured, and immutable data structure used in applications like digital currencies. We discussed how blocks are cryptographically linked together to ensure that old records can not be changed, and why these types of databases are useful for applications where the immutability of data is key.

But how does this work technically? Let's take a deeper look at how blockchains are secured by the application of hashing and proof-of-work.

It's all in ~~your~~ the block's head

The *block header*

The key to understanding blockchain lies in a data structure included in every block of every blockchain. This data structure is called the *block header*, and it contains several critical bits of information needed to secure the chain as it grows.

Let's look at the Bitcoin block header as an example. Each Bitcoin block contains an 80 byte header with the following information:

Version - the software version of the Bitcoin protocol
Timestamp - expressed in seconds since the Unix Epoch
Merkle Root - for our purposes here, we'll say this is a "fingerprint" of all the transactions in this block
Difficulty target - A 256 bit integer used in calculating proof of work
Nonce - The value added to the block header to demonstrate proof of work
Previous block hash - The SHA-256 hash of the previous block header

All of this data is important and serves a purpose in the Bitcoin protocol. However, I've highlighted the `previous block hash` because it is particularly important when discussing how the blockchain is secured!

A quick review of hashes

Before we discuss the critical role that the `previous block hash` section of the block header plays in securing the blockchain, let's step back and recall what a hash function does. When "hashing" data, a special algorithm called a "hash function" takes the data and outputs a unique "fingerprint" of the input data. These functions (at least if they are implemented properly) have two very important properties.

First, a particular chunk of data *always* produces the same hash (or "fingerprint") every time it is run through the function. If you run `Hello` through the SHA-256 hashing algorithm, the result will *always* be `185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969`.

Second, good hash functions avoid *collisions*, where different data results in the same hash output. Using a proven algorithm such as SHA-256 means that for every different input, a different hash is produced. Even if a single bit of input changes, the hash is radically different. "Hello" will produce a very different output than "hello", even though only a few bits of the input are changed.

Hashes inside hashes inside hashes

Understanding these properties of hashes, we can better understand the interesting approach that blockchains take to securing the integrity of data in previous blocks when combined with *proof of work*.

Each time a block is generated, proof of work is generated in the form of the `nonce` included in the block header. This is computationally intensive and essentially proves that a miner spent a good bit of CPU power to find the answer to this cryptographic puzzle.

Now, when the nonce is included in the block header with the other data including the *previous block hash*, we can hash the *entire block header* to generate the unique fingerprint. This "block hash" is unique, and changing *any data whatsoever* in the block header would create a radically different block hash.

Okay, so each block has an associated hash. What's the big deal? How does that help secure the blockchain? The magic of blockchain lies in that critical piece of data known as the `previous block hash`. Recall that changing any bit of data in the input radically changes the output of a hash function. So what would happen if we tried to change a transaction 2 blocks back in our blockchain?

If a node tries to broadcast a fake blockchain to the network with a fake transaction 2 blocks back, the block hashes for each subsequent block would be *radically different.*

Let's look at an example. Let's say we have a really simple blockchain with some transaction data like so (Note - these hashes are made up for demonstration purposes): `Block 2: Time - 3000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 5345245 Prev block hash - 33a0b89fcce723e9f41f5d756ab1c20584afbe6dfa9ea18838ff3caf0915b5f5 Transaction: Bob pays Alice 6 units Block 1: Time - 2000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 2356343 Prev block hash - f4ebb8b56f590188f5824276af552cd51a48ba774e3ad1350c2800b116d8f6f5 Transaction: Alice pays Bob 5 units Block 0: Time - 1000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 1232341234 Prev block hash -`

`0000000000000000000000000000000000000000000000000000000000000000 Transaction: Alice pays Bob 1 unit`

Now let's say Bob gets greedy and tries to say that Alice paid him 10 units in the first transaction: `Block 2: Time - 3000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 9987983 Prev block hash - 9ae343e333cbb96427eb333bb8c443359e3cf926c9de9845ceb583577b945afb Transaction: Bob pays Alice 6 units Block 1: Time - 2000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 390970 Prev block hash - 3f82f4cfe059b5a69a0fd5b4d34774af5ecdc672d988320d5fd186998969a645 Transaction: Alice pays Bob 5 units Block 0: Time - 1000 Difficulty - 1000000000000000000000000000000000000000000000000000000000 Nonce - 235235 Prev block hash - 0000000000000000000000000000000000000000000000000000000000000000 Transaction: Alice pays Bob 10 units`

Notice how different the hashes are for blocks 1 and 2 in Bob's fake blockchain. If these hashes are to be considered valid by a node in this currency's network, then each block *must also demonstrate proof of work*. Since the data has changed in a block, a new nonce must be found to show that work was done.

Here is the most critical part - since the hash for block 0 has changed and is included in block 1, proof of work has to be re-done for block 1. And since block 1's hash is included in block 2, proof of work has to be re-done for block 2. In other words, to try and fake a transaction 2 blocks back, Bob has to re-do proof of work for 3 whole blocks!! In the meantime, legitimate nodes only have to try and find a solution for the current block. *It is clearly impractical, if not impossible, to "fake" a blockchain more than one or two block old, because proof of work has to be redone for many blocks in the time the legitimate network only has to prove one.*

Faking a blockchain take too much work!

Due to the interesting combination of hashing and proof-of-work algorithms, it is incredibly difficult if not impossible to fake history in a blockchain database. Because each block contains the hash of the previous block, changing history blocks back means that every bit of the chain on forward must be forged. While legitimate nodes only have to prove work for one block in that span of time, an attacker would have to calculate for many. Unless a malicious party has some amount of computing power the rest of us don't know about, it's nearly impossible to do so.

For the extra curious, Satoshi covers the math behind this concept extensively in section 11 of the [Bitcoin whitepaper]. While I don't claim to understand this math very well myself, the paper does a good job of explaining its conclusions that forging blockchain history is a fool's errand.