Implementing proof-of-work (Javascript blockchain, part 2)

In my previous blog post we created a simple blockchain in Javascript to demonstrate how a blockchain works. However many people commented and said that the implementation wasn't complete and that they could still fool the system. They are correct! Our blockchain needs another mechanism to secure itself against attacks. So let's take a look at how we can do that!

This blog post is part of a whole series:

- Part 1: Implementing a basic blockchain
- **Part 2: Implementing proof-of-work**
- Part 3: Transactions & mining rewards

The problem

Right now we can create blocks and add them to our blockchain really quickly. And this creates 3 problems:

- First of all: people can create blocks incredibly fast and **spam** our blockchain. A flood of blocks would overload our blockchain and would make it unusable.

- Secondly, because it's so easy to create a valid block, people could **tamper** with one of the blocks in our chain and then simply recalculate all the hashes for the blocks after it. They would end up with a valid blockchain, even though they've tampered with it.
- And thirdly, you can combine the two problems above to effectively **take control** of the blockchain. Blockchains are powered by a peer-to-peer network in which the nodes will add blocks to the longest chain available. So you can tamper with a block, recalculate all the other blocks and then add as many blocks as you want. You will then end up with the longest chain and all the peers will accept it and start adding their own blocks to it.

Clearly we need a solution for these problems. Enter: proof-of-work.

What is proof-of-work?

Proof-of-work is a mechanism that existed before the first blockchain was created. It's a simple technique that prevents abuse by requiring a certain amount of computing work. That amount of work is key to prevent spam and tampering. Spamming is no longer worth it if it requires a lot of computing power.

Bitcoin implements proof-of-work by requiring that the hash of a block starts with a specific number of zero's. This is also called the difficulty.

But hang on a minute! How can the hash of a block change? In case of Bitcoin a block contains details about a financial transaction. We sure don't want to mess with that data just to get a correct hash!

To fix this problem, blockchains add a `nonce` value. This is a number that gets incremented until a good hash is found. And because you cannot predict the output of a hash function, you simply have to try a lot of combinations before you get a hash that satisfies the difficulty. Looking for a valid hash (to create a new block) is also called "mining" in the cryptoworld.

In case of Bitcoin, the proof-of-work mechanism ensures that only 1 block can be added every 10 minutes. You can imagine spammers having a hard time to fool the network if they need so much compute power just to create a new block, let alone tamper with the entire chain.

Implementing proof-of-work

So how do you implement it? Let's start by modifying our block class and adding the `nonce` variable in it's constructor. I'll initialize it's value and set it 0.

```
constructor(index, timestamp, data, previousHash = '') {
    this.index = index;
    this.previousHash = previousHash;
    this.timestamp = timestamp;
    this.data = data;
    this.hash = this.calculateHash();
    this.nonce = 0;
}
```

We also need a new method that will increment the nonce until we get a valid hash. Again, this is dictated by the difficulty, so we'll receive the difficulty as a parameter:

```
mineBlock(difficulty) {
    while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0"))
        this.nonce++;
        this.hash = this.calculateHash();
    }
    console.log("BLOCK MINED: " + this.hash);
}
```

And finally we also need to change the `calculateHash()` function because right now it doesn't use the nonce variable to calculate the hash.

```
calculateHash() {
    return SHA256(this.index +
        this.previousHash +
        this.timestamp +
        JSON.stringify(this.data) +
        this.nonce
    ).toString();
}
```

If you throw it all together you get a Block class that looks like this:

```
class Block {
    constructor(index, timestamp, data, previousHash = '') {
        this.index = index;
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.data = data;
        this.hash = this.calculateHash();
        this.nonce = 0;
    }

    calculateHash() {
        return SHA256(this.index + this.previousHash + this.timestamp + JSON.string
    }

    mineBlock(difficulty) {
        while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0
            this.nonce++;
            this.hash = this.calculateHash();
        }
        console.log("BLOCK MINED: " + this.hash);
    }
}
```

Blockchain modifications

Now that our blocks have a nonce and can be mined, we need to make sure that our blockchain supports this new behaviour as well. Let's start by adding a new property to our blockchain to keep track of the difficulty of the chain. I'll start by setting it to 2 (meaning that the hashes of blocks should start with 2 zero's).

```
constructor() {
    this.chain = [this.createGenesisBlock()];
    this.difficulty = 2;
}
```

All that's left now is to change the `addBlock()` method so that it actually mines the block before adding it to our chain. Here we'll pass the difficulty to the block:

```
addBlock(newBlock) {
    newBlock.previousHash = this.getLatestBlock().hash;
    newBlock.mineBlock(this.difficulty);
    this.chain.push(newBlock);
}
```

And that's it! Our blockchain now has proof-of-work and protection against spam and attempts to tamper with it.

Testing it

Let's now test our blockchain and see what effects proof-of-work has on adding new blocks to our chain. I'll use the same code as before. We'll start by creating a new instance of our blockchain and then adding 2 simple blocks to it.

```
let savjeeCoin = new Blockchain();

console.log('Mining block 1');
savjeeCoin.addBlock(new Block(1, "20/07/2017", { amount: 4 }));

console.log('Mining block 2');
savjeeCoin.addBlock(new Block(2, "20/07/2017", { amount: 8 }));
```

If you run this, you'll see that adding new blocks is still very fast. That's because the difficulty is only set to 2 (and because computers are really fast).

If you increase the difficulty to 5, you'll see that a modern computer takes about 10 seconds to mine a block. Increase it further and you'll have a great protection from attackers.

Disclaimer

Just as before a quick warning: this is by no means a complete blockchain implementation. It still lacks many features (like a P2P network). This is just to show how blockchains work internally.

Also: mining blocks in Javascript is not really fast because it just uses a single thread.

Next up

- Part 1: Implementing a basic blockchain
- **Part 2: Implementing proof-of-work**
- Part 3: Transactions & mining rewards

Conclusion & source code

Proof-of-work is essential for the security and integrity of blockchains. Without it, we couldn't trust it enough to store information it.

The source code of this project is available on Github.

Posted on 03 Sep 2017

6 Comments **Savjee.be** 1 **Login**

Recommend 5 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Michal Boleslav Měchura • 3 months ago

Great article. I've a quick question to verify that I've understood things correctly. Should you not also update the blockchain's isChainValid() method to verify that each block in the chain has the correct difficulty (= that the hash starts with the correct number of zeros)? Without that, the proof-of-work seems useless to me because it is never looked at.

3 • Reply • Share ›

Anuj Gupta • 2 months ago

Anuj Gupta • 2 months ago

Great article Savjee !!
Would love to see an article of how to setup P2P network so this chain can be viewed but not tampered by anyone running a blockchain node.

^ | ˅ • Reply • Share ›

Pramesh Bajracharya • 3 months ago

Great article. We would love to see you work more on this. A kinda series may be??

^ | ˅ • Reply • Share ›

Comments continue after advertisement

Drive now - Terms Apply

Earn at least $2,000 in San Francisco for your first 200 Trips.
Limited time guarantee! Terms apply.

Learn More

Sponsored by **Uber**

Report ad

vzool • 3 months ago

Hi,

Would you please explain Difficulty?

Thanks

^ | ˅ • Reply • Share ›

b3lun ➦ vzool • 2 months ago

"Bitcoin implements proof-of-work by requiring that the hash of a block starts with a specific number of zero's. This is also called the difficulty."

what is there more to explain?

^ | ˅ • Reply • Share ›

vzool ➦ b3lun • 2 months ago

Yes, of course ...

I understand how mining works, I'm talk about another connected subject.
Explain how the P2P Network shares & agree on one "Difficulty Value"?

^ | ˅ • Reply • Share ›

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Privacy**