Build a simple Blockchain in R

I have read a great joke recently:

“A boy asked his bitcoin investing dad for 1 bitcoin for his birthday.

Dad: What? 15.554 $ ??? 14.354 $ is a lot of money! What do you need 16.782 $ for anyway?”

Everybody is talking about bitcoins and cryptocurrencies now. Everybody! Much more interesting whether there is a BitCoin bubble or not is the question how this technology actually works. I first got introduced to *the Blockchain* at a meeting of the Blockchain Society at Oxford (Yes, there is a Blockchain Society). Everybody there seemed very excited about the potential of this new technology. And to be fair, the promise of the blockchain sounds intriguing: A decentralised, incorruptible database of monetary transactions (BitCoins), contracts (SmartContracts) or whatever you like ().

While the basic idea is quite intuitive the question how blockchains actually work on a technical level is a bit harder to understand. Last week I came across this article on R bloggers, where “BigData Doc” build a Blockchain entirely in R. A blockchain built in R might not be the most efficient and practical thing in the world, but it is a great way to understand the programming principles behind it: “If you can code it, you certainly understand it”.

Let’s do it. Let’s build a very simple blockchain with very basic programming in R:

**What is a blockchain?**

First, we store some data (e.g. financial transactions) in a container – which is called a block. If we now take another block we can add them together to form a chain – the blockchain.

**1. Blocks**

How does a block look like? At the very least we need some data and a timestamp. We will also add an index and a self-identifying hash. We will just store it in a basic list for a moment.

Before we can start building the blockchain – aka chaining different containers with data together – we need to get to know two more concepts: *Hashing* and the *Proof-of-Work-Algorithm.*

**2. Hash**

A hash helps to ensure the integrity of a block by connecting it to the other blocks in the chain. A hash function takes something as an input and gives us a unique output. How does this help us? “Hash functions are often used for proving that something is the same as something else, without revealing the information beforehand.” In our case we not only input information about the block (index, timestamp, data) to the hash function, but also the hash of the previous block. This means we can only check whether a block is valid if we know the hash of the previous block, which is created using the hash of the block before and so on and so forth. This provides us with an immutable, sequential chain of blocks.

*Code here*

**3. Proof-of-Work**

If there is a lot of information that must be stored in the blockchain we will need to create a lot of new blocks. In many cases we want to control how many new blocks are created. In the case of cryptocurrencies for example coins would lose their value if there is an infinite amount of coins that can be created every second.

Therefore, we add a so-called “Proof-of-Work” (PoW) algorithm which controls the difficulty of creating a new block. Proof means that the computer has performed a certain amount of work. In practice the goal is to create an item that is hard to create but easy to verify. I our example we will use the following “task” as a PoW: *find the next number that is divisible by 99 and divisable by the proof-number of the last block.*

*Code here*

For blockchains like BitCoin or Ethereum the job of creating new blocks is done by so called *miners*. When a new block has to be created, the miner which solves the PoW problem the fastest creates the new block and gets rewarded (e.g. through fees on transactions).

**4. Genesis Block**

Now we know how a block looks like, how blocks are chained together using hashes and how the pace of creating new blocks is being regulated by PoWs.

Before we start adding data to our blockchain we need to start the chain somewhere. This is done using a so-called *Genesis Block.* It contains no data and arbitrary values for proof and previous hash (as there is no previous block).

*Code here*

**5. Building the Blockchain**

Now we can start building the blockchain. We start with the genesis block and then add 20 blocks using a loop.

*Code here*

We see that the first blocks are created quite quickly, but as the proof of work gets much harder for every new block, it takes increasingly more time.

In this little blog post we created the tiniest blockchain. The main goal was to introduce how a blockchain looks like and introduce some of the core concepts behind it. To put a blockchain into production much more work is needed (and probably not in R): Set up an API, create wallets, etc.

I oriented myself for this little introduction on the blog post of Gerald Nash (<https://medium.com/crypto-currently/lets-build-the-tiniest-blockchain-e70965a248b>) who implemented the tiniest blockchain in Python and I added the PoW Python implementation from [Daniel van Flymen](https://hackernoon.com/@vanflymen?source=post_header_lockup) (<https://hackernoon.com/learn-blockchains-by-building-one-117428612f46>) .

To dive deeper into the topic of blockchains I recommend the orginial paper (<https://bitcoin.org/bitcoin.pdf>) and the blog post from “Bigdata Doc” (<https://www.r-bloggers.com/building-your-own-blockchain-in-r/>).