←Home

Building Blockchain in Go. Part 1: Basic Prototype

AUGUST 16, 2017
GOLANG BLOCKCHAIN BITCOIN

> Chinese translations: by liuchengxu, by zhangli1.

Introduction

Blockchain is one of the most revolutionary technologies of the 21st century, which is still maturing and which potential is not fully realized yet. In its essence, blockchain is just a distributed database of records. But what makes it unique is that it's not a private database, but a public one, i.e. everyone who uses it has a full or partial copy of it. And a new record can be added only with a consent of other keepers of the database. Also, it's blockchain that made cryptocurrencies and smart contracts possible.

In this series of articles we'll build a simplified cryptocurrency that's based on a simple blockchain implementation.

Block

Let's start with the "block" part of "blockchain". In blockchain it's blocks that store valuable information. For example, bitcoin blocks store transactions, the essence of any cryptocurrency. Besides this, a block contains some technical information, like its version,

current timestamp and the hash of the previous block.
In this article we're not going to implement the block as it's described in blockchain or Bitcoin specifications, instead we'll use a simplified version of it, which contains only significant information. Here's what it looks like:

```
type Block struct {
        Timestamp     int64
        Data          []byte
        PrevBlockHash []byte
        Hash          []byte
}
```

`Timestamp` is the current timestamp (when the block is created), `Data` is the actual valuable information containing in the block, `PrevBlockHash` stores the hash of the previous block, and `Hash` is the hash of the block. In Bitcoint specification `Timestamp`, `PrevBlockHash`, and `Hash` are block headers, which form a separate data structure, and transactions (`Data` in our case) is a separate data structure. So we're mixing them here for simplicity.

So how do we calculate the hashes? The way hashes are calculates is very important feature of blockchain, and it's this feature that makes blockchain secure. The thing is that calculating a hash is a computationally difficult operation, it takes some time even on fast computers (that's why people buy powerful GPUs to mine Bitcoin). This is an intentional architectural design, which makes adding new blocks difficult, thus preventing their modification after they're added. We'll discuss and implement this mechanism in a future article.

For now, we'll just take block fields, concatenate them, and calculate a SHA-256 hash on the concatenated combination. Let's do this in `SetHash` method:

```
func (b *Block) SetHash() {
        timestamp := []byte(strconv.FormatInt(b.Timestamp, 10))
        headers := bytes.Join([][]byte{b.PrevBlockHash, b.Data,
timestamp}, []byte{})
        hash := sha256.Sum256(headers)

        b.Hash = hash[:]
}
```

Next, following a Golang convention, we'll implement a function that'll simplify the creation of a block:

```
func NewBlock(data string, prevBlockHash []byte) *Block {
        block := &Block{time.Now().Unix(), []byte(data),
prevBlockHash, []byte{}}
        block.SetHash()
        return block
}
```

And that's it for the block!

Blockchain

Now let's implement a blockchain. In its essence blockchain is just a database with certain structure: it's an ordered, back-linked list. Which means that blocks are stored in the insertion order and that each block is linked to the previous one. This structure allows to quickly get the latest block in a chain and to (efficiently) get a block by its hash.

In Golang this structure can be implemented by using an array and a map: the array would keep ordered hashes (arrays are ordered in Go), and the map would keep `hash → block` pairs (maps are unordered). But for our blockchain prototype we'll just use an array, because we don't need to get blocks by their hash for now.

```
type Blockchain struct {
        blocks []*Block
}
```

This is our first blockchain! I've never thought it would be so easy 😉

Now let's make it possible to add blocks to it:

```
func (bc *Blockchain) AddBlock(data string) {
        prevBlock := bc.blocks[len(bc.blocks)-1]
        newBlock := NewBlock(data, prevBlock.Hash)
```

```
	bc.blocks = append(bc.blocks, newBlock)
}
```

That’s it! Or not?..

To add a new block we need an existing block, but there’re not blocks in our blockchain! So, in any blockchain, there must be at least one block, and such block, the first in the chain, is called genesis block. Let’s implement a method that creates such a block:

```
func NewGenesisBlock() *Block {
	return NewBlock("Genesis Block", []byte{})
}
```

Now, we can implement a function that creates a blockchain with the genesis block:

```
func NewBlockchain() *Blockchain {
	return &Blockchain{[]*Block{NewGenesisBlock()}}
}
```

Let’s check that the blockchain works correctly:

```
func main() {
	bc := NewBlockchain()

	bc.AddBlock("Send 1 BTC to Ivan")
	bc.AddBlock("Send 2 more BTC to Ivan")

	for _, block := range bc.blocks {
		fmt.Printf("Prev. hash: %x\n", block.PrevBlockHash)
		fmt.Printf("Data: %s\n", block.Data)
		fmt.Printf("Hash: %x\n", block.Hash)
		fmt.Println()
	}
}
```

Output:

```
Prev. hash:
Data: Genesis Block
Hash:
aff955a50dc6cd2abfe81b8849eab15f99ed1dc333d38487024223b5fe0f1168

Prev. hash:
aff955a50dc6cd2abfe81b8849eab15f99ed1dc333d38487024223b5fe0f1168
Data: Send 1 BTC to Ivan
Hash:
d75ce22a840abb9b4e8fc3b60767c4ba3f46a0432d3ea15b71aef9fde6a314e1

Prev. hash:
d75ce22a840abb9b4e8fc3b60767c4ba3f46a0432d3ea15b71aef9fde6a314e1
Data: Send 2 more BTC to Ivan
Hash:
561237522bb7fcfbccbc6fe0e98bbbde7427ffe01c6fb223f7562288ca2295d1
```

That’s it!

Conclusion

We built a very simple blockchain prototype: it’s just an array of blocks, with each block having a connection to the previous one. The actual blockchain is much more complex though. In our blockchain adding new blocks is easy and fast, but in real blockchain adding new blocks requires some work: one has to perform some heavy computations before getting a permission to add block (this mechanism is called Proof-of-Work). Also, blockchain is a distributed database that has no single decision maker. Thus, a new block must be confirmed and approved by other participants of the network (this mechanism is called consensus). And there’re no transactions in our blockchain yet!

In future articles we’ll cover each of these features.

Links:

1. Full source codes: https://github.com/Jeiwan/blockchain_go/tree/part_1
2. Block hashing algorithm: https://en.bitcoin.it/wiki/Block_hashing_algorithm

Ivan Kuznetsov
Write things

tweet Share

Read more

What is Lightning Network and How to Try It Today	Mar 2 2018
Building Blockchain in Go. Part 7: Network	Oct 6 2017
Building Blockchain in Go. Part 6: Transactions 2	Sep 18 2017
Building Blockchain in Go. Part 5: Addresses	Sep 11 2017
Building Blockchain in Go. Part 4: Transactions 1	Sep 4 2017
Building Blockchain in Go. Part 3: Persistence and CLI	Aug 29 2017
Building Blockchain in Go. Part 2: Proof-of-Work	Aug 22 2017
TIL: Convolutional Filters Are Weights	Aug 5 2017